P 1677/85

mikroKLAN 13

1

1985

informatyka

BASIC dla mikrokomputerów

PASCAL MT+

Superkomputery CRAY

WYDAWNICTWO SZKOLNE

Cena 100 zł

informatyka

1 1985

Nr 1
Miesięcznik Rok XX
Styczeń 1985

Organ Komitetu Informatyki MNSzWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

KOLEGIUM REDAKCYJNE:

Mgr inż. Zbigniew GLUZA, dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (zastępca redaktora naczelnego), prof. dr hab. Leon ŁUKASZEWICZ (redaktor naczelny), mgr inż. Andrzej J. PIOTROWSKI, dr inż. Janusz ZALEWSKI

STALE WSPÓŁPRACUJĄ:

Mgr inż. Witold ABRAMOWICZ (Szwajcaria), mgr inż. Ryszard K. KOTT (Wielka Brytania), dr Jacek OWCZARCZYK, dr Andrzej SZAŁAS, dr Jakub TATARKIEWICZ, mgr inż. Teresa WILCZEK

PRZEWODNICZĄCY RADY PROGRAMOWEJ:

Prof. dr hab. Tadeusz PECHE

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". Zam. 2345. Obj. 4,0 ark. druk. Nakład 6500 egz. N-28.

ISSN 0542-9951

Cena egzemplarza zł 100,—
Prenumerata roczna zł 1200,—

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA CZASOPISM I KSIĄŻEK TECHNICZNYCH
SIGMA
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

BASIC dla mikrokomputerów — ewolucja języka
Ryszard K. Kott 2

PASCAL MT+ — dialekt PASCALA dla mikrokomputerów
Artur Krępski 6

PLZ — język programowania mikroprocesorów (2)
Wiesław Wilczyński 9

Projekt normy IEEE — arytmetyka zmiennoprzecinkowa
Zbigniew Banasik 12

mikroKLAN 13

— Norma GKS
— Przystawka do ZX81
— Mikrokomputer MERITUM-I
— FORTH (5)
— Akademia mikroKLANU (2): silnia

SAMOTESTY 23

— II/B. Reprezentacja, przepływ i przekształcanie danych

Z KRAJU 24

— Uchwała I Walnego Zjazdu PTI w sprawie rozwoju informatyki w Polsce
— Wynik z komputera

ZE ŚWIATA 26

— Świat superkomputerów CRAY

RECENZJE 30

— Zastosowanie mikroprocesorów w energoelektronice

TERMINOLOGIA 30

— Słownik terminów związanych z arytmetyką zmiennoprzecinkową

CZWARTA OKŁADKA — *Rafał Pietrak, Jakub Tatarkiewicz*

W NAJBLIŻSZYCH NUMERACH:

- Waldemar Kapuścik o projektowaniu baz danych
- Konrad Jabłoński o języku BCPL
- Wacław Iszkowski o mechanizmach komunikacji
- Marek Dziedzic, Krzysztof Perycz i Jerzy Wiliński o emulatorze E6RM-E
- Andrzej I. Litwiniuk o kompilatorze LOGLANU dla MERY 400
- Zbigniew Kierzkowski i Jacek Małuszyński o współbieżnej aktualizacji bazy danych
- Janusz Zalewski o tendencjach w dziedzinie komputerowych systemów sterowania w 1983 roku
- Julian Winiewski o języku C
- Frank Land o systemach informacyjnych
- Wojciech Cellary i Jan Węglarz o nauczaniu mikroinformatyki

POLITECHNIKA ŚLĄSKA Biblioteka Główna w Gliwicach
P. 1877/85

Problem perspektywy

Wkraczamy oto w dwudziesty rok INFORMATYKI. Nie czas na jubileuszową fetę, więc rocznicę potraktujmy jako okazję do krytycznej autorefleksji. Okazja to tym lepsza, że postawieni jesteśmy przed koniecznością wyboru dalszej drogi.

Informatyka światowa przeżywa okres niezwykłego boomu. Do nas przenika zaułkami, chyłkiem. „Oficjalny" park komputerowy wykrusza się coraz bardziej — wszystkie statystyki biją na alarm. „Nieoficjalnie" zaś, prywatnie, rzemieślniczo, polonijnie, półzachodnio ta informatyczna feeria zaczyna pojawiać się i u nas. Stanęliśmy jakby na rozdrożu dwóch informatyk — różniących się od siebie bardziej sposobem funkcjonowania informatyków, niż technologią.

INFORMATYKA jest również w wewnętrznym konflikcie. Z jednej strony — pojawia się coraz więcej entuzjastów, których wyobraźnię i zapał rozbudziły dostępne już ich kieszeni mikrokomputery; a wraz z nimi — wiele tematów i spraw żądających natychmiastowego umieszczenia na łamach. Z drugiej — zachowana jest dotychczasowa organizacja pisma.

Dzisiejsza INFORMATYKA — zważywszy jak wiele dziedzin i tematów obejmuje swym zasięgiem (nie mając wszak konkurencji!) — powinna być pismem znacznie obszerniejszym, z liczniejszą redakcją, mogącą wykonywać różne czynności — i redaktorskie, i organizacyjne. Towarzysząc dziedzinie będącej napędem rozwoju gospodarczego, nie może się ukazywać w tak prymitywnej szacie edytorskiej i tak niskim nakładzie.

Nakład jest zaskakująco mały — choćby w zestawieniu z liczebnością informatycznego środowiska. Pismo — podobnie zresztą jak inne czasopisma naukowo-techniczne — sprzedawane jest niemal w całości w prenumeracie, Wydawca nie chce podjąć ryzyka skierowania go do kiosków. W ten sposób pismo pozostaje nadal na granicy dochodowości, co z kolei wytrąca nam argumenty za jego rozwojem. Koło się więc zamyka — decyzję może podjąć tylko Wydawnictwo, zupełnie do tego nieskore.

Objętość jest zaniżona — nawet w stosunku do tej, jaką mieliśmy do 1983 roku. Wówczas — uzasadniając decyzję trudnościami poligraficznymi i niedostatkiem papieru — zmniejszono łamy INFORMATYKI o osiem stron. To „tymczasowe ograniczenie na najbliższy rok" zostało teraz przedłużone — na obecny rok. A w zasadzie — do czasu wybudowania przez SIGMĘ własnej drukarni... I do tej pory będziemy na łasce „obcych" drukarń. 32 strony + 4 strony okładki to objętość niezbyt poważna, jeśli całą dziedzinę za poważną uznać.

Warunki narzucone przez drukarnie nie ulegają poprawie, a nawet nieco się pogorszyły — w efekcie braku elementarnych choćby inwestycji. Autorzy dziwią się czasem, że nie możemy w artykułach stosować kilku rodzajów wyróżnień (np. kursywa, ciemniejszy druk, podkreślenia), bo przecież wszędzie się je stosuje, szczególnie w zapisach „formalnych". To zrozumiałe zdziwienie, ale niestety — nic się nie da zrobić. Drukarze dzisiaj nie znoszą trudnego składu, więc przy każdej okazji przypominają o możliwości usunięcia INFORMATYKI z drukarni. Co gorsze, niektóre pisma już usunęli...

Redakcja etatowa liczy trzy osoby, w tym dwie zajmujące się stroną merytoryczną. Taka ekipa może podołać obowiązkom miesięcznika, którego życie płynie niespiesznie, leniwie, gdzie niemal wszystko jest przewidywalne i przewidziane. Gdy rytm dziedziny, której się towarzyszy, jest szybki, nie można sobie pozwolić na niedotrzymywanie kroku. Więcej — trzeba być w czołówce, bo inaczej nie jest się wcale.

Podniesiono cenę pisma. W istocie nie mamy na to żadnego wpływu, więc nie będziemy przepraszać. Wydawnictwo zakomunikowało nam, że wzrost cen wynika ze znacznego podwyższenia kosztów poligrafii i papieru oraz... ze zbyt niskiego nakładu.

Przy tych wszystkich ograniczeniach i zakazach, pojawiła się m o ż l i w o ś ć, którą witamy z satysfakcją. Otóż począwszy od tego roku będziemy mogli wydawać niewielkie monograficzne broszury — sprzedawane w kioskach. Być może taka forma zrekompensuje nieco niedostatek łamów pisma. W broszurach tych zamierzamy przedstawić: wprowadzenie do mikroinformatyki, kompletne konstrukcje mikrokomputerów, zbiór programów użytkowych w BASICU, pełne opisy wybranych mikroprocesorów. Szczegóły — niebawem.

Już sama lista powyższych tematów świadczy chyba o wyborze, jakiego powoli dokonujemy. Mówiąc oględnie, staramy się śledzić w informatyce to wszystko, co w niej żywe, co jest świadectwem ruchu. Nie będziemy sekundować muzealnym zabytkom sztuki konstruktorskiej, niezależnie od tego jak długo będą reanimowane. Za punkt odniesienia musimy przyjąć informatykę światową, jeśli nie chcemy ślepo brnąć do całkowitego i nieodwracalnego rozkładu społecznego, bo przecież na gospodarczym się nie kończy.

INFORMATYKA jest teraz jakby połączeniem dwóch bytów — pisma obejmującego całą dziedzinę oraz mikroKLANU — miejsca dla amatorów mikroinformatyki (co niekoniecznie znaczy informatyków-amatorów). Byty te mają wprawdzie wspólne cechy, nie na tyle jednak, by ich symbioza była dostatecznie uzasadniona. Oba też mają szersze aspiracje i przeszkadzają sobie nawzajem — zabierając cenne miejsce. Przygotowujemy się do wydzielenia mikroKLANU i utworzenia zeń osobnego czasopisma; w ramach obecnej, rozszerzonej redakcji. Na przekór opisanej niemocy, ale w zgodzie z obiektywną koniecznością. Wspomniane broszury byłyby dobrym wstępem (i zachętą) dla nowego miesięcznika.

Wszystkie organizacje informatyczne, a także pojedynczych informatyków prosimy o wsparcie takiej inicjatywy, choćby w formie rozgłaszanego postulatu...

Rok 1985, rok dwudziestolecia, będzie zapewne dla INFORMATYKI rokiem przełomowym. Jak głęboki będzie to przełom, zależy także od Czytelników. Od presji, jaką na nas wywrzecie, zwiększając determinację — niezbędną w administracyjnych potyczkach.

ZBIGNIEW GLUZA

Oprogramowanie mikrokomputerów

Blokiem artykułów dotyczących oprogramowania mikrokomputerów, zgrupowanych w poprzednim (12/1984) i bieżącym numerze INFORMATYKI, kontynuujemy prezentację tej tematyki, rozpoczętą w lipcu ubiegłego roku.

Sądzę, że treści poszczególnych artykułów nie potrzeba zbyt szeroko komentować. Niewątpliwie najsilniej reprezentowane są w nich języki programowania. Dwa najpopularniejsze w programowaniu mikroprocesorów — BASIC i PASCAL — z pewnością zainteresują większość Czytelników. Każdy praktykujący programista musi znać oba te języki, choćby ze szkoły. Choć PASCAL jest znormalizowany przez ISO (norma ISO 7185), a BASIC w wersji minimalnej przez ANSI (obecnie trwają prace nad jego rozszerzeniem — raport X3J2 84/26), oba mają za małe możliwości, aby zadowolić wszystkich użytkowników. Liczba dialektów tych języków sięga już z pewnością kilkudziesięciu. Ich zasięg jest stokroć większy niż prezentowanych uprzednio — FORTHA i PL/M, warto więc obserwować dalszą ewolucję.

Motywem przedstawienia innego języka — PLZ — jest coraz powszechniejsze stosowanie w Polsce mikroprocesorów Z80, choćby w mikrokomputerach osobistych ZX firmy SINCLAIR.

Cykl programowania, sprowadzający się u nas do redagowania, tłumaczenia i uruchamiania programów, ilustrują w części artykuły poprzedniego numeru, a nieco więcej światła na środowisko, w jakim powinien pracować programista, rzuca opis PASCALA MT+.

Najmniej informacji można zdobyć na temat stosowanych w Polsce mikrokomputerowych systemów operacyjnych, gdyż — jak nietrudno stwierdzić — prawie ich się nie używa. Z tego względu do następnego numeru włączymy przedruk wywiadu z Gary Kildallem, pożyteczny o tyle, że ilustruje anatomię działania firmy, która osiągnęła sukces na rynku oprogramowania. Podczas gdy sami nie wyszliśmy jeszcze w tej dziedzinie poza etap raczkowania, na świecie powstają wyroby zaliczane u nas do zjawisk nieznanych. Już obecnie jądro systemu operacyjnego iRMX86 firmy INTEL jest zakodowane w mikroukładzie (tzw. operating system firmware chip — 80130), podobnie jak — system CP/M (mikroukład 80150, por. INFORMATYKA, nr 2/1984, str. 19).

Innym zagadnieniem właściwie u nas nie poruszanym jest przenośność oprogramowania między różnymi systemami operacyjnymi. W dużym stopniu zapewniają ją języki wysokiego poziomu, ułatwiając jednolite wyrażanie programów źródłowych, ale każdy system operacyjny ma specyfizne dyrektywy, które uniemożliwiają wzajemną wymienność programów. W celu stworzenia podstaw do pełnej zgodności systemów operacyjnych pod tym względem, powstał projekt normy IEEE P855 MOSI (ang. Microprocessor Operating System Interface). Jej celem jest opracowanie standardowych zasad wywoływania funkcji systemu operacyjnego przez programy użytkowe. Być może będzie to przydatne nie tylko do wymiany programów między systemami operacyjnymi różnych producentów, ale stworzy możliwość opracowania programu pod kontrolą jednego systemu operacyjnego (le-

RYSZARD K. KOTT
Instytut Informatyki
Politechnika Warszawska

BASIC dla mikrokomputerów — ewolucja języka

BASIC jest od dawna jednym z popularniejszych języków programowania, szczególnie wśród początkujących programistów i amatorów. W ostatnim czasie jego popularność jeszcze bardziej wzrosła w efekcie rozpowszechnienia się mikrokomputerów. Mimo znanych wad tego języka, takich jak ubogie struktury danych, brak lokalności zmiennych czy brak instrukcji strukturalnych, jego prostota i łatwość operowania ciągami znaków (ang. strings) zjednują mu wciąż nowych zwolenników. Oczywiście naprzeciw życzeniom użytkowników wychodzą producenci i dla każdego komputera osobistego (ang. personal computer) dostępny jest translator lub interpreter BASICA. Dla wielu systemów mikrokomputerowych BASIC jest jedynym językiem programowania sprzedawanym wraz ze sprzętem. Zwiększa to wymagania dotyczące funkcji języka, co przyczynia się do jego szybkiej ewolucji. Jak więc BASIC wygląda dziś?

Materiału porównawczego dostarcza artykuł T. Li [5], w którym zestawiono instrukcje BASICA dla sześciu tanich i najpopularniejszych w USA w 1980 roku komputerów osobistych. Komputery te działają w oparciu o pamięć ROM i nie mają pamięci zewnętrznych w postaci dysków (COMMODORE PET, APPLE II, RADIO SHACK TRS 80, ATARI 400 i 800, TEXAS INSTRUMENT TI 99/4 oraz EXIDY SORCERER). Omówione instrukcje można uporządkować w grupy tematyczne, co pozwala na wyciągnięcie ciekawych wniosków dotyczących ewolucji języka w nowym środowisku. Wnioski te w pełni potwierdza analiza BASICA dla najpopularniejszych obecnie w Polsce mikrokomputerów osobistych, tj. modeli ZX81 oraz ZX SPECTRUM produkowanych przez firmę SINCLAIR (W. Brytania).

PODSTAWOWY ZBIÓR INSTRUKCJI

Omówienie podstawowego zbioru instrukcji języka nastręcza nieco kłopotów i wymaga komentarza. Od 1978 roku istnieje norma amerykańska [1], obejmująca jednak tylko minimalny zbiór konstrukcji językowych. Praktycznie każda realizacja BASICA zawiera rozszerzenia w stosunku do tej normy [2, 3]. Ostatnio, pod koniec 1982, opublikowano nową propozycję standardu, omówioną już na łamach INFORMATYKI [4]. W podsumowaniu tego omówienia wyrażono jednak — podzielaną przeze mnie — wątpliwość, czy język ten można jeszcze nazwać BASICEM. Propozycja opisuje bowiem język programowania współbieżnego, zawierający instrukcje strukturalne, procedury z parametrami i złożone struktury danych, daleko odbiegający od tego, co przyzwyczailiśmy się rozumieć pod nazwą BASIC. Proponowany standard nie będzie więc podstawą do porównania.

Poniższą listę podstawowych instrukcji języka zestawiłem głównie w celach poglądowych. Dążyłem jednak do tego, aby stanowiła ona wspólny podzbiór popularnych wersji języka (włączono do niej też standard z roku 1978).

Podstawowe elementy tego zbioru można podzielić na następujące grupy:

- instrukcje czynne: LET, STOP, GOTO, ON..GOTO, IF.. THEN, FOR..TO..STEP, NEXT, GOSUB, RETURN, RESTORE, INPUT, PRINT, READ, WRITE, ON..ERROR, PRINT-USING, CHANGE, DELIMIT
- instrukcje bierne: REM, DIM, DATA, BASE, DEFFN, FILE, instrukcja formatu

piej dostosowanego do tych zadań), a wykonania go — pod kontrolą drugiego (np. XENIX i iRMX dla mikroprocesorów INTEL, UNIX i RSX-11 dla mikrokomputerów firmy DEC).

Konieczność zapewnienia przenośności oprogramowania zmusza producentów i użytkowników do wspólnych działań na rzecz normalizacji. Wyrazem tych tendencji w zakresie oprogramowania numerycznego jest omówiony w tym numerze projekt normy IEEE P754, dotyczącej reprezentacji liczb zmiennoprzecinkowych. Określa ona wymagania i opcje na format argumentów i wyników operacji. Choć w chwili pisania artykułu przedstawiony projekt nie był jeszcze ostatecznie zatwierdzony (w 1984 r. opracowano już dziesiątą wersję projektu normy IEEE, natomiast ósma wersja została przyjęta w 1982 r. jako norma IEC 559 — Binary Floating Point Arithmetic for Microprocessor Systems), według jego zaleceń zrealizowano już wiele procesorów zmiennoprzecinkowych, zarówno jednoukładowych (np. INTEL 8087, MOTOROLA 68881, a także procesory firmy ZILOG i NATIONAL), jak i złożonych z kilku mikroukładów. Obecnie prowadzi się prace nad normą dotyczącą reprezentacji liczb zmiennoprzecinkowych, niezależnej od podstawy (ang. radix independent floating-point, P854).

Normalizuje się także postać kodu wynikowego programów. Projekt normy zwanej MUFOM (ang. Microprocessor Universal Format for Object Modules) stanowi przedmiot prac grupy P695 IEEE. Niezależne prace w tym kierunku, rozpoczęte już wcześniej, doprowadziły nawet do powstania kilku norm wewnętrznych, jak CUFOM (ang. CERN Universal Format for Object Modules) w laboratoriach jądrowych CERN, pod Genewą, i OMF (ang. Object Module Format) firmy INTEL.

Czy można zidentyfikować nas na tej mapie? W Polsce prowadzi się wiele prac nad oprogramowaniem mikrokomputerów, ale ich ilość nie może przerodzić się w jakość, bo za wysoka jest bariera organizacyjna. Pomijam względy techniczne, których jest znacznie więcej niż można pomieścić w tak krótkim omówieniu.

Trudno doszukać się oznak świadczących o rozwoju oprogramowania mikrokomputerów w Polsce, w sposób przekraczający rozmiarami romantyczną zabawę na miarę mikroKLANU, gdzie entuzjaści wymieniają adresy. Świat ucieka nam nie tylko w produkcji mikroukładów, co najmniej tak samo szybko oddalamy się od poziomu światowego w dziedzinie wytwarzania oprogramowania. Jaka jest więc moja diagnoza? Tak jak w sferze sprzętu nie zrobimy ani kroku do przodu bez znaczących inwestycji w produkcję układów scalonych, tak też w dziedzinie oprogramowania mikrokomputerów potrzebne są odpowiednie posunięcia na szczeblu decyzyjnym, aby przeciwdziałać rozproszeniu prac. Do tego czasu pozostaje nam jedynie zabawa w programistów-amatorów.

JANUSZ ZALEWSKI

- funkcje arytmetyczne: ABS, ATN, COS, SIN, TAN, MOD, EXP, LOG, SOR, RND, INT
- funkcje dotyczące operacji napisowych i funkcje pomocnicze: CHR$, LEN, STR$, CAL$, STR, VAL, POS, TAB.

ROZSZERZENIA JĘZYKA DLA MIKROKOMPUTERÓW

W rozwoju BASICA dla zastosowań mikrokomputerowych można wyróżnić kilka wyraźnych tendencji.

Przede wszystkim dąży się do ulepszenia samego języka, tj. do zwiększenia jego możliwości, oraz do wprowadzenia lepszych konstrukcji językowych. Te dążenia nie są specyficzne dla zastosowań w środowisku mikrokomputerowym, są raczej odbiciem aktualnych poglądów na inżynierię oprogramowania. Nową propozycję standardu można prawie w całości zaliczyć do tego kierunku.

Inne tendencje wypływają wprost z nowych zastosowań. Wszystkie więc porównywane w [5] implementacje BASICA zawierają instrukcje pozwalające na bezpośredni dostęp z poziomu języka do zasobów fizycznych maszyny (komórek pamięci o adresach bezwzględnych, portów itd.).

Wprowadzono też do języka wiele nowych instrukcji związanych z obsługą kolorowych monitorów graficznych, w które wyposaża się mikrokomputery.

Basic jest często jedynym narzędziem komunikacji operatora z maszyną, dlatego włącza się do niego także operacje systemowe, jak np. manipulowanie plikami. Warto też zauważyć, że dla różnych mikrokomputerów dyrektywy systemowe, których nie włączono do języka, również wykazują dużo podobieństw. Oczywiście, wymienione w poszczególnych punktach instrukcje, funkcje czy dyrektywy nie zawsze są dostępne we wszystkich porównywanych mikrokomputerach.

Nowe konstrukcje językowe

Ta grupa rozszerzeń zbioru podstawowego, często bardzo śmiałych (np. instrukcja WAIT), została praktycznie w całości objęta nową propozycją normy. Szczególnie dotyczy to grupy instrukcji strukturalnych i instrukcji synchronizacji procesów.

- **Instrukcje strukturalne i sterujące:**

IF..THEN instrukcja — prosta instrukcja IF
IF..THEN..ELSE... — pełna instrukcja IF
IF..GOSUB..., RETURN — warunkuje wejście do (lub wyjście z) podprogramu
ON..GOSUB..., RETURN — przełącznik
RESUME... — wznowienie wykonania programu po wystąpieniu błędu i obsłudze (od wskazanej linii)
WAIT... — synchronizacja z procesem zewnętrznym (wstrzymanie wykonania programu do spełnienia podanych warunków).

- **Deklaracje** (opisane rozróżnianie typów danych nie zostało włączone do nowej propozycji normy):

nazwy — wieloznakowe
DIM... — definiowanie tablic napisów
DEF FN... — definiowanie funkcji wieloargumentowych i wieloliniowych
DEFINT ... — definiowanie zmiennych, o nazwach zaczynających się od określonej litery — jako liczb całkowitych
DEFDBL ... — j.w. — jako liczb podwójnej precyzji
WEFSNG ... — j.w. — jako liczb pojedynczej precyzji
DEFSTR ... — j.w. — jako zmiennych napisowych.

- **Funkcje arytmetyczne:**

FIX (wyr.) — część całkowita wyrażenia
ERL (wyr.) — nr linii, w której wystąpił błąd
CLOG (wyr.) — logarytm o podstawie 10
CINT (wyr.)
CDBL (wyr.) — konwersja wyrażenia od odpowiedniej postaci
CSNG (wyr.).

- **Operatory logiczne** (przewidziane w nowym standardzie): AND, OR, NOT.

- **Funkcje działające na napisach i funkcje pomocnicze** (wyraźne jest dążenie do wzmocnienia tej, i tak silnej, strony języka, jaką jest przetwarzanie znakowe; dotyczy to szczególnie wycinków napisów):

ASC (napis) — udostępnia kod ASCII pierwszego znaku napisu
CODE (napis) — jak wyżej
STRING (n, znak) — udostępnia napis o długości n utworzony z podanych znaków
LEFT$ (napis, n) — udostępnia lewostronny wycinek napisu o długości n (od początku)
RIGHT$ (napis, n) — udostępnia prawostronny wycinek napisu o długości n (od końca)
MID$ (napis p, n) — udostępnia wycinek napisu o długości n i początku p.
Ponadto w tablicach napisów można indeksować pojedyncze znaki.

Zapewnienie dostępu do zasobów fizycznych maszyny

Wymienione w tym punkcie instrukcje i funkcje są bardzo typowe dla mikrokomputerów, natomiast nie są objęte nową propozycją normy.

Przede wszystkim wprowadzono do języka adresowanie bezwzględne w kodzie wynikowym, tj. w pamięci operacyjnej, z poziomu programu. Nie należy więc mylić wymienionej niżej instrukcji CALL (skok do procedury w kodzie maszynowym, o znanym adresie początkowym) z właściwą instrukcją obsługi podprogramów GOSUB (skok do podprogramu o znanym początkowym numerze linii).

Zapewniono także dostęp do specyficznych urządzeń wejścia-wyjścia spotykanych w mikrokomputerach, a szczególnie do tzw. portów (identyfikowanych tak, jak to się czyni z poziomu asemblera). Można oczekiwać dalszych rozszerzeń w tej grupie instrukcji, gdyż w mikrokomputerach służącym celom rozrywkowym na pewno pojawią się nowe, efektowne przystawki (np. muzyka, sterowanie zdalne czy rozpoznawanie mowy).

● **Adresowanie bezwzględne:**
PEEK (adres) — udostępnia wartość zapamiętaną w komórce (bajcie) o podanym adresie (funkcja)
POKE adres, val — zapamiętuje podaną wartość pod podanym adresem (instrukcja)
CALL adres — powoduje skok do podprogramu w języku maszynowym, o podanym adresie (podprogram powinien się kończyć instrukcją maszynową RET)
USR (adres) — j.w., ale jest to funkcja; wynik umieszcza we wskazanych rejestrach mikroprocesora
VARPTR (nazwa) — udostępnia adres zmiennej o podanej nazwie
FRE(x), ERE(x$) — podaje ilość pamięci dostępnej jeszcze do zapamiętywania zmiennych zwykłych lub napisowych.

● **Specyficzne operacje wejścia-wyjścia:**
IN (port) — pobiera wartość z portu (funkcja)
OUT port, wyr. — wysyła wartość do portu (instrukcja)
GET, INKEY$, CALL KEY — pobiera znak z klawiatury
PRINT — wydruk znaku we wskazanym miejscu ekranu
LPRINT — skierowanie wydruku na drukarkę (a nie na ekran)
CALL SOUND — ustala wyjście przystawki dźwiękowej
CALL JOYSTIC — sprawdza port (wejście) manipulatora (dźwigni)
PADDLE — pobiera wartość z wejścia zatrzaskowego
TI wyr. — ustala licznik czasu rzeczywistego
WAIT — oczekiwanie na akcję operatora.

Instrukcje obsługi monitora graficznego

Wraz z zastosowaniem monitorów wprowadzono wiele instrukcji związanych z programowaniem obrazu i struktur graficznych. Rozwój sprzętu wywołał liczne problemy związane z oprogramowaniem monitorów kolorowych. Znalazło to odbicie w BASICU dla mikrokomputerów, który — dzięki takim instrukcjom, jak np. CLS, CALL COLOR, DRAWTO — umożliwia programowanie kolorowych obrazów.

● **Włączanie i wyłączanie trybu współpracy z ekranem:**
GRAPHIC — włączenie trybu grafiki; wybór strony grafiki
TEXT — przejście z trybu graficznego do znakowego.

● **Obsługa koloru:**
CLS — kasowanie obrazu
CALL COLOR — definiowanie tła dla poszczególnych znaków
CALL SCREEN — wybranie koloru tła ekranu
COLOR=n — ustawienie koloru do następnego punktu.

● **Obsługa pojedynczych punktów matrycy ekranu:**
POINT — sprawdzenie określonej komórki „wideo"
PLOT, SET (x,y) — zapalenie punktu na pozycji (x,y)
RESET (x,y) — ustawienie kursora w pozycji (x,y).

● **Obsługa linii:**
HLIN..AT — rysowanie poziomej linii o danym numerze
VLIN..AT — rysowanie pionowej linii w danej kolumnie
VTAB(x) — przesunięcie kursora o x linii w dół
DRAWTO — rysowanie linii od ostatniego punktu do określonej pozycji.

● **Instrukcje specjalne:**
SPEED=wyrażenie — określa szybkość przesyłania znaku na ekran.

Instrukcje dotyczące operacji na plikach

Ważne miejsce w ewolucji BASICA zajmują operacje na plikach. Umożliwiają one definiowanie urządzeń zewnętrznych jako plików i operacje na nich, jak również organizują współpracę z taśmami i kasetami, np. instrukcje OPEN, STORE, UPDATE. Możliwości BASICA dla mikrokomputerów zostały rozszerzone przez wprowadzenie — jako instrukcji — operacji systemowych na plikach. Są to — na przykład — operacje edytora AUTO, DELETE, EDIT. Instrukcje te pozwalają na redagowanie programu z poziomu BASICA:
APPEND — dodaje dane do końca pliku danych
CLOSE — zamyka urządzenie jako plik
OPEN — otwiera urządzenie jako plik wejściowy lub wyjściowy
POP — usuwa wynik edycji ze stosu
STORE — zapisuje wartości tablicy na kasetę
UPDATE — odczytuje zawartości pliku z taśmy i zapisuje poprawioną zawartość.

Typowe dyrektywy systemowe

Istnieje coraz wyraźniejsza tendencja do włączania dyrektyw do języka BASIC jako instrukcji. Jest to spowodowane możliwościami wynikającymi z zastosowania interpreterów BASICA w komputerach osobistych.

● **Dyrektywy edytora:**
AUTO m,n — automatyczne numerowanie linii
CLOAD — ładowanie tekstu programu z kasety
CSAVE — składowanie programu na kasecie
CLOAD? — weryfikacja programu na kasecie
DELETE m — usunięcie linii
EDIT m — poprawienie linii
LIST m,n — wydrukowanie programu.

● **Dyrektywy monitora:**
NEW — usunięcie programu i zerowanie wskaźników
RUN — uruchomienie programu BASIC
TRON, TROF — włączenie (wyłączenie) śledzenia wykonania
BREAK m — ustawienie pułapki
UNBREAK — usunięcie pułapki
STOP — zatrzymanie programu
CONTINUE — kontynuacja po zatrzymaniu.

● **Dyrektywy dostępu do systemu:**
HIMEN, LOMEN — ustalenie maksymalnego lub minimalnego dostępnego adresu
SYSTEM — wykonanie programu w języku maszynowym.

BASIC DLA MIKROKOMPUTERÓW ZX81 I ZX SPECTRUM

Najpopularniejszymi obecnie w Polsce mikrokomputerami osobistymi są ZX81 oraz ZX SPECTRUM firmy SINCLAIR. Ponieważ są to konstrukcje późniejsze w stosunku do omawianych w [5] (SPECTRUM pochodzi z roku 1982), stanowią one dobre uzupełnienie dla niniejszych rozważań. Zarysowane w poprzednim punkcie tendencje ewolucji języka BASIC zdecydowanie potwierdzają się także w tym przypadku.

Oba mikrokomputery mają bardzo podobne odmiany języka. Ponieważ ZX81 jest lepiej znany, omówię pokrótce BASIC ZX SPECTRUM, który jest znacznie ciekawszy, co wynika przede wszystkim z dostępności grafiki i koloru (dla ZX81 przewidziano tylko czarno-biały i alfanumeryczny monitor TV). Ponadto pamięć 16 K ROM w SPECTRUM umożliwia zrealizowanie bogatszego funkcjonalnie systemu interpretera.

Nowe konstrukcje językowe

Zrezygnowano z utrzymania zasady, że w jednym wierszu musi znajdować się jedna instrukcja. Instrukcja może zajmować wiele wierszy, a jeden wiersz może zawierać wiele instrukcji. Numer wiersza podaje się tylko na początku, a dwukropek jest separatorem instrukcji.

W instrukcji IF, po słowie THEN można podać całą listę instrukcji, co ułatwia nadanie programowi dobrej struktury. Operatory AND, OR i NOT mogą działać nie tylko na argumentach zerojedynkowych, lecz także na dowolnych wyrażeniach numerycznych i napisowych. Funkcje arytmetyczne występują w bogatym zestawie, łącznie ze

stałą funkcją PI. Podobnie obszerna jest grupa funkcji i operacji napisowych, dla których przewidziano nowy, wygodny element składniowy do wskazywania dowolnych wycinków.

Dostęp do zasobów fizycznych maszyny

Adresowanie bezwzględne zapewniają: instrukcja POKE oraz funkcja odczytu PEEK i adresowania (skoku) — USR. Zadawanie wartości dwójkowych ułatwia funkcja BIN. Ponieważ w firmowym podręczniku języka BASIC dla tego komputera opisano wszystkie zmienne systemowe, to powyższe instrukcje umożliwiają odczytanie dowolnych informacji systemowych, a także ich zapis, tj. zmianę parametrów systemu!

Specyficzne operacje wejścia-wyjścia reprezentują: odwołania do portów mikrokomputera — IN oraz OUT, do klawiatury — funkcja INKEY$, do określonego punktu ekranu — słowo AT w instrukcji PRINT, do wbudowanego głośniczka — instrukcje BEEP i PAUSE, do drukarki — instrukcje LPRINT, LLIST i COPY (ta ostatnia — do kopiowania obrazu na drukarkę firmową).

Ponadto do synchronizacji programu za pomocą klawiatury może służyć instrukcja PAUSE.

Osługa kolorowego monitora graficznego

Monitorem ekranowym dla mikrokomputera SPECTRUM może być zwykły telewizor czarno-biały lub kolorowy w systemie PAL (SECAM daje obraz czarno-biały). Na ekranie można wyświetlać znaki albo pojedyncze punkty obrazu (ang. pixel). Matryca znaku składa się z 8×8 punktów. Wyświetlane 22+2 linie po 32 znaki w ośmiu kolorach (włącznie z białym i czarnym). Drukować można w sposób zwykły, tzn. „czarne" na „białym" (TRUE VIDEO) albo odwrócony — INVERTED VIDEO. Obraz można skasować instrukcją CLS.

Dostęp do poszczególnych znaków ekranu zapewniają: instrukcja PRINT z frazą AT, funkcja odczytu SCREEN$ oraz funkcja badania atrybutów pola znakowego ATTR. Oprócz bogatego zestawu znaków alfanumerycznych (m.in.: $, £, #, c, {, [, ~, |, /) można drukować tzw. znaki graficzne. Jest to szesnaście kombinacji powstałych z podziału matrycy znakowej na ćwiartki. Ponadto użytkownik może zdefiniować 22 własne znaki, zadając matrycę 8×8 przy użyciu funkcji USR i BIN oraz instrukcji POKE.

Do obsługi grafiki (176×256 punktów) przewidziano następujące instrukcje: pisania punktu — PLOT, kreślenia odcinków i łuków — DRAW, rysowania okręgów — CIRCLE, a także funkcję badania stanu punktu — POINT.

Silną stroną opisywanego systemu jest to, że rysowanie punktów nie wyklucza pisania znaków — i na odwrót. Ponadto wszędzie można uzyskiwać efekty kolorystyczne, np. czerwony okrąg na jasnoniebieskim tle czy zielone litery na żółtym tle.

Trzeba przyznać, że do operowania kolorem znaleziono bardzo eleganckie rozwiązanie składniowe, zgodne z duchem języka BASIC. Kolor oraz efekty dodatkowe opisuje się za pomocą szeregu elementów składniowych. Można je uwzględniać w dowolnej instrukcji odwołującej się do ekranu, poczynając od zwykłej instrukcji PRINT (nawet bez frazy AT). W jednej instrukcji może być kilka elementów dodatkowych, a ich pominięcie powoduje przyjęcie wartości domniemanych — zazwyczaj odpowiednio: czarnego dla znaków (punktów), białego dla tła.

Do określenia jednego z ośmiu kolorów służą frazy: INK — podaje kolor „atramentu", PAPER — kolor tła, BORDER — kolor marginesu otaczającego tło. Dodatkowe efekty są sterowane frazami: BRIGHT — jaskrawość, FLASH — błyskanie, INVERSE — inwersja na kolory dopełniające, OVER — zapisywanie na tle w sposób widoczny (kolor kontrastowy).

Instrukcje i dyrektywy systemowe

W ZX SPECTRUM nie ma rozróżnienia pomiędzy instrukcjami języka BASIC a dyrektywami systemu — wszystkie są obsługiwane przez wspólny interpreter. W rezultacie dowolną instrukcję można napisać bez numeru, tj. w trybie natychmiastowej interpretacji, a każdą dyrektywę można wstawić do programu jako instrukcję. W pierwszym przypadku korzysta się z systemu jak z wygodnego kalkulatora o olbrzymiej, symbolicznej pamięci, lub — jak z monitora umożliwiającego śledzenie programu. W drugim przypadku istnieje możliwość niebanalnego programowania systemowego (jak np. ładowanie i wykonywanie innych programów).

Instrukcje (dyrektywy) redagowania i operowania plikami

Wprowadzanie programu z klawiatury przebiega zgodnie z konwencją, w której instrukcja poprzedzona numerem jest instrukcją do zapamiętania przez edytor. Istnieje pojęcie linii bieżącej, którą można poprawiać po przywołaniu do bufora klawiatury klawiszem EDIT. Tekst programu (w postaci zakodowanej interpretera) można składować na taśmie kasetowej dyrektywą SAVE i załadować ponownie dyrektywą LOAD. Można ponadto sprawdzić poprawność zapisu na kasecie (VERIFY) oraz dokonywać łączenia tekstu kilku programów lub plików (MERGE). Firma oferuje również niestandardowe urządzenie szybkiej kasetowej pamięci magnetycznej o nazwie MICRODRIVE (15 kB/s). Do obsługi plików na tym nośniku przewidziano cały zestaw dodatkowych instrukcji (OPEN, CLOSE, CAT, DELETE, ERASE, MOVE).

Dyrektywy (instrukcje) systemowe

Program można uruchomić od wskazanego miejsca lub od początku dyrektywą RUN, a zatrzymać (nie zawsze) — dyrektywą STOP. Do przerwania programu we wszystkich pozostałych przypadkach służy dyrektywa, a właściwie klawisz — BREAK. Program zatrzymany można wznowić dyrektywą CONTINUE. Nie ma specjalnych dyrektyw do śledzenia programu, bo nie jest to konieczne. Pułapki można wstawiać umieszczając w dowolnym miejscu program dyrektywą STOP (dopuszcza się wiele instrukcji w wierszu). Zerowania zmiennych i kasowania obrazu dokonuje się za pomocą dyrektywy CLEAR, natomiast program można całkowicie usunąć z pamięci dyrektywą NEW.

Dyrektywy do obsługi taśmy kasetowej (LOAD, SAVE i VERIFY) odnoszą się nie tylko do programów w języku BASIC. Przy ich użyciu można ładować, składować i weryfikować zawartość dowolnych plików. Dla wygody, w składni uwzględniono szereg słów do określania rodzaju pliku:
SCREEN$ — dla plików zawierających obraz ekranu TV
CODE — dla pliku z kodem (ciągiem bajtów)
DATA — dla pliku z danymi liczbowymi
DATA$ — dla pliku z danymi tekstowymi
LINE — dla plików tekstowych, ale z wykonaniem programu od wskazanego miejsca.

* * *

Przegląd różnych implementacji języka BASIC dla mikrokomputerów osobistych pozwala na zorientowanie się w wyraźnych tendencjach ewolucji języka. Oprócz zmian wynikających z innego obecnie rozumienia roli języków programowania, ujawniły się tendencje mające związek z nowym środowiskiem — „niszą ekologiczną". Polegają one na uwzględnieniu mechanizmów dostępu do fizycznych zasobów maszyny, gwałtownym rozwoju instrukcji związanych z oprogramowaniem kolorowych monitorów graficznych i różnych specyficznych przystawek oraz — na zacieraniu się różnicy między instrukcjami języka a dyrektywami systemu.

Należy przypuszczać, że podobne tendencje powinny występować dla każdego języka programowania używanego w podobnej roli, tzn. jako główne albo jedyne narzędzie komunikacji człowiek-maszyna w środowisku mikrokomputerów. Tendencje te powinny być bardzo zbliżone dla języków opartych na interpreterach, natomiast w dużym stopniu podobne dla języków mających translatory (różnica wynika z możliwości włączania dyrektyw do języka — bardzo naturalnej dla interpreterów).

LITERATURA

[1] American National Standards Institute: American National Standard for Minimal BASIC. Standard ANSI X3.60, 1978
[2] Czech Z., Nałęcki K., Wołek S.: Programowanie w języku BASIC. WNT, Warszawa, 1977
[3] Iszkowski W., Maniecki M.: Programowanie w języku BASIC. PWN, Warszawa, 1980
[4] Iszkowski W., Maniecki M.: Standaryzacja języka BASIC. Informatyka, nr 5, 1983
[5] Li T.: Whose BASIC Does What? Byte, pp. 318—327, January 1981.

ARTUR KRĘPSKI
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

PASCAL MT+ dialekt PASCALA dla mikrokomputerów

Język PASCAL MT+ najłatwiej scharakteryzować przez narzędzia ułatwiające posługiwanie się nim — system SPP i kompilator.

System SPP (ang. Speed Programming Package) jest zestawem bardzo prostych, a jednocześnie wygodnych i wydajnych narzędzi wspomagających pracę użytkownika języka PASCAL MT+.

Kompilator, oprócz tego, że akceptuje pełny standard PASCALA, zawiera tak wiele różnego rodzaju rozszerzeń i narzędzi, że stwarza pełny dostęp do całości zasobów mikrokomputera i systemu operacyjnego, eliminując całkowicie potrzebę używania języka asemblera.

System SPP razem z kompilatorem PASCAL MT+ stanowi bardzo efektywne narzędzie do programowania szerokiej klasy problemów na mikrokomputerach o ograniczonej konfiguracji. Jest on dostępny pod nadzorem systemów operacyjnych CP/M i MP/M dla komputerów zbudowanych na bazie mikroprocesorów INTEL 8080 i Z80, wyposażonych w pamięć operacyjną o pojemności od 52 do 64 KB, klawiaturę i monitor ekranowy, a także — jeden napęd dyskietek. Dokumentacja — w postaci krótkiego (70 stron) opisu SPP dla użytkownika [2] oraz 200-stronicowego podręcznika PASCALA MT+ [3] — zawiera informacje wystarczające do instalowania (a nawet modyfikowania), użytkowania i uczenia systemu SPP.

Stosunkowo duży koszt systemu SPP (łącznie z kompilatorem PASCAL MT+ — ok. 500 dol.) spowodowany jest dużym popytem i brakiem liczącego się konkurenta. System rozpowszechnia firma DIGITAL RESEARCH (USA), a aktualną liczbę instalacji ocenia się już na tysiące. Można przewidywać [4] dalsze intensywne rozpowszechnianie się systemu SPP i duży rozwój związanego z nim oprogramowania użytkowego.

CHARAKTERYSTYKA SYSTEMU SPP

Użytkownik steruje działaniem systemu SPP i poszczególnymi narzędziami wybierając jedną z kilku możliwości wyszczególnionych na ekranie monitora w postaci tzw. menu (ang. screen oriented menu-driven) — rys. 1.

```
Speed Programming Package

Options:        E)dit
                R)eformat
                S)yntax check
                V)ariable check
                X)eq
                D)ir
                L)ink
                F)ast compile
                Q)uit

Command?
```

Rys. 1. Menu systemu SPP

System SPP zawiera następujące narzędzia programowania: edytor (E), program formatowania tekstów pascalowych (R), program zestawiania uporządkowanej alfabetycznie listy identyfikatorów w programie pascalowym (V), analizator składniowy (S) oraz kompilator (F) języka PASCAL MT+.

Istnieje ponadto możliwość korzystania z następujących poleceń systemu operacyjnego bez konieczności opuszczania systemu SPP: wyświetlanie na ekranie monitora indeksu plików na określonej dyskietce (D), połączenie (L) i wykonanie (X) programu w PASCALU. Polecenie Q służy do opuszczenia systemu SPP i powrotu do systemu operacyjnego.

Zasadą działania systemu SPP jest operowanie od razu na całości tekstu pascalowego. Na początku działania systemu tekst ten jest wczytywany z pliku źródłowego do specjalnego bufora w pamięci operacyjnej, natomiast na końcu — jest on wpisywany na ten sam plik (poprzednia wersja tekstu jest przy tym automatycznie kopiowana na dodatkowy plik).

Edytor

Edytor systemu SPP służy do konwersacyjnego operowania tekstem źródłowym (znajdującym się w całości w buforze w pamięci operacyjnej) za pomocą poleceń uaktywnianych kluczem sterującym (CNTRL) i klawiszem o jednoliterowym kodzie.

Ekran monitora służy jako „okienko" przesuwające się po tekście źródłowym — wszelkie zmiany na ekranie są automatycznie wprowadzane do tekstu w buforze.

Edytor działa w jednym z trzech trybów: wstawiania, zamieniania i usuwania tekstu. Może on operować całymi liniami, słowami (ciągami znaków oddzielonych spacjami) oraz pojedynczymi znakami.

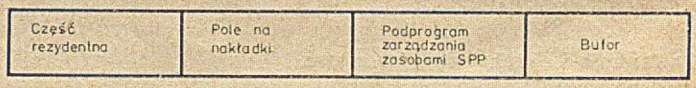
Dostępne są polecenia przesuwania kursora po ekranie (we wszystkich kierunkach, na początek i koniec ekranu) i po całym tekście (liniami i stronami o wielokrotności rozmiarów ekranu, na początek i koniec tekstu, do linii o określonym numerze).

Specjalne polecenie (z własnym menu) umożliwia wyświetlanie na ekranie zawartości dowolnego pliku i jej wstawienie na określone miejsce w operowanym tekście, zaznaczenie początkowej i końcowej linii bloku tekstu źródłowego i wpisanie go w określony plik, a ponadto — dostarcza informacji o stanie wykorzystania zasobów systemu SPP (np. zajętość bufora czy długość tekstu mierzoną w liniach).

Dodatkowe polecenia służą do wyszukiwania i zamieniania określonych napisów — ciągów znaków (automatycznie w całym tekście, określona liczbą razy lub pojedynczo z każdorazowym potwierdzeniem przez użytkownika), adiustację tekstu, wstawianie pustych linii, kopiowanie z przenoszeniem całych fragmentów tekstu źródłowego.

Właściwości funkcjonalne systemu

System SPP jest zakodowany w języku PASCAL w postaci jednego programu podzielonego na część rezydentną (stale przechowywaną w pamięci operacyjnej) oraz dziewięć nakładek (ładowanych do pamięci w przypadku odwołania się do nich) realizujących poszczególne narzędzia. Część rezydentna zajmuje 6 KB pamięci, natomiast nakładki łącznie 103 KB (głównie kompilator PASCAL MT+).



Rys. 2. Schemat podziału pamięci dla systemu SPP

Długość bufora przeznaczona na tekst źródłowy zależy od konfiguracji sprzętu i systemu operacyjnego. Wynosi ona od 7 KB (np. dla systemu MP/M) do 19 KB (system CP/M w minimalnej konfiguracji i 64 KB pamięci operacyjnej).

Stosunkowo duże ograniczenia długości tekstu źródłowego przetwarzanego przez system SPP wymuszają na użytkowniku podział programu na rozsądnie małe segmenty, co może skłaniać również do logicznej modularyzacji programu. Z drugiej strony — redagowanie tekstu w całości znajdującego się w pamięci operacyjnej (przy stosunkowo wolnej pracy urządzeń wejścia-wyjścia) jest bardzo efektywne. Odpowiedź systemu na większość poleceń (nawet związanych z wyszukiwaniem ciągów znaków) jest szybka, niemal natychmiastowa.

KOMPILATOR

Kompilator PASCAL MT+ akceptuje pełny język PASCAL zgodny z normą ISO [1]. Nieistotne odstępstwa od normy są następujące:
— niedozwolona jest instrukcja skoku w procedurach rekurencyjnych
— następuje zignorowanie błędu w instrukcji case, gdy wartość wyrażenia wyboru nie odpowiada żadnej z etykiet wyboru
— poprawna wartość funkcji eof dla pliku nietekstowego występuje tylko wówczas, gdy ostatni element pliku jest jednocześnie na końcu sektora dyskietki.

PASCAL MT+ rozszerzono o wiele konstrukcji dodatkowych (nie zawsze jednak zgodnie z duchem języka i regułami semantycznymi normy). Celem tych rozszerzeń jest udostępnianie zasobów mikrokomputera i systemu operacyjnego (co w praktyce pozwala unikać użytkowania języka asemblera), lub też dążenie do utworzenia konstrukcji umożliwiających efektywniejsze kodowanie, niż można tego dokonać stosując tylko język standardowy.

Poniżej zestawiono najważniejsze rozszerzenia (niektórych użyto w przykładzie), bardziej zwracając uwagę na rodzaj mechanizmu językowego niż na jego opis.

- Wprowadzono pojęcie modułu (blok ujęty w nawiasy **module** i **modend**) i umożliwiono oddzielną kompilację. Wszystkie obiekty globalne modułu są automatycznie eksportowane, a w celu importu — ich typ należy poprzedzić słowem external, np.:
var i, j, k: **external** integer;
external procedure p(x: real);
- Możliwe jest opisanie struktury nakładkowej programu (dodatkowa konwencja zadawania grupy nakładkowej procedury w nawiasach kwadratowych przy deklaracji procedury) oraz dynamiczne ładowanie procedury w czasie wykonania programu (ang. chaining).
- Wprowadzono dodatkową instrukcję exit — wyjście z określonej procedury lub funkcji, a także procedurę whlt — wyjście do systemu operacyjnego.
- Wprowadzono dodatkowe typy standardowe: byte i word, a także standardową funkcję konwersji wrd (wartości dowolnego typu porządkowego na typ word).
- Wprowadzono dodatkowy typ standardowy string (np. **var** s: string [50]) wraz z całym zestawem standardowych procedur i funkcji.
- Wprowadzono dodatkowe funkcje i procedury — bez żadnych gwarancji bezpiecznego używania — służące do operacji na bitach, bajtach i wektorach bajtów (znaków). Odpowiadają one zwykle pojedynczym rozkazom maszynowym (są więc bardzo efektywne).
- Wprowadzono dodatkowe procedury i funkcje operowania plikami udostępniające bezpośrednio w PASCALU pliki o dostępnie swobodnym — odpowiadające operacjom na zbiorach danych w systemie operacyjnym (m.in. otwieranie, zamykanie, badanie wyników operacji na pliku), a także wykonujące wczytywanie i zapisywanie wartości w formacie szesnastkowym — odpowiadające operacjom kanałowym.
- Wprowadzono procedury i funkcje (m.in. addr, sizeof) udostępniające atrybuty obiektów programu pascalowego.
- Umożliwiono zapisywanie stałych w programie w notacji ósemkowej i szesnastkowej.
- Wprowadzono operacje logiczne na całych słowach (a nie tylko na wartościach logicznych).
- Rozszerzono składnię instrukcji case — po słowie else można określić instrukcję, którą należy wykonać zawsze wtedy, gdy wartość wyrażenia wyboru nie odpowiada żadnej z wartości etykiet wyboru.

Dodatkowe narzędzia programowania

Kompilator PASCAL MT+ ma wbudowane narzędzia, których celem jest ułatwienie pracy przy tworzeniu programów.

- Parametry kompilacji zadawane w instrukcjach stanowiących rozszerzenie komentarza, np. (* $E+,X+ *), służą do sterowania procesem kompilacji, np. w celu zwiększenia efektywności kodu wynikowego, włączenia sprawdzeń generowanych przez kompilator czy warunkowej kompilacji.
- Parametry kompilatora wyszczególniane w poleceniu do systemu operacyjnego uaktywniającym kompilator (mtplus), np.:

MTPLUS A:TEST.SRC $ B PX A;

służą opisaniu zasobów systemu w czasie działania kompilatora.

- Program łączący, który łączy do 32 plików wynikowych (w tym także modułów wybieranych z bibliotek) tworząc jeden moduł ładowalny, umożliwia zadawanie struktury nakładkowej zarówno kodu programu, jak i segmentów danych.
- Dekoder programów wynikowych na postać zbliżoną do asemblera.

Kompilator PASCAL MT+ pozwala dokonywać wstawek w języku asemblera (przygotowano do tego specjalną, prostą składnię rozpoznawaną przez kompilator) za pomocą dodatkowej instrukcji inline.

- Bibliotekarz służy dwóm celom:
— przekształcaniu sekwencyjnych plików wynikowych w bibliotekę modułów ładowalnych (np. PASLIB)
— przekształcaniu wynikowych plików tworzonych przez kompilator PASCAL MT+ na format akceptowany przez inne programy łączące (np. L80 czy Link80).
- Program uruchomieniowy w postaci oddzielnej nakładki kompilatora PASCAL MT+ umożliwia wyświetlanie wartości zmiennych i wyświetlanie zawartości tablicy symboli (nazw procedur i nazw zmiennych), wykonywanie programu pojedynczymi instrukcjami połączone z ich trasowaniem (w postaci numeru linii liczonych osobno w każdym module), wykonywanie programu od początku lub zadanego punktu wstrzymania (można również kasować zdefiniowane uprzednio punkty wstrzymania), modyfikowanie zawartości pamięci oraz instruowanie użytkownika. Korzystanie z tego programu wymaga kompilacji programu pascalowego z włączonym parametrem D.

Właściwości kompilatora

Kompilator PASCAL MT+ jest dwuprzebiegowy (I przebieg — analiza składniowa, II przebieg — generowanie kodu), składa się z części rezydentnej i sześciu nakładek — zajmując łącznie 92 KB. Wymaga 36 KB na kod kompilatora w pamięci operacyjnej oraz przynajmniej 7 KB na tablicę symboli (maksymalnie 17 KB), a ponadto — pliku roboczego na kod pośredni kompilowanego programu (ok. 1/3 rozmiaru pliku źródłowego). W sumie, zaleca się przynajmniej 48 KB pamięci operacyjnej (oprócz pamięci na system operacyjny) na kompilowanie średnich programów pascalowych.

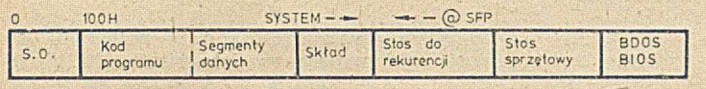
Program łączący (nie jest nakładkowany) wymaga 11 KB pamięci operacyjnej na swój kod, a całą resztę dostępnej pamięci zużywa na tablicę symboli i kod łączonych programów. Faza łączenia wymaga dostępu do systemowych plików zawierających moduły wynikowe dołączane do programu wygenerowanego przez kompilator. Koniecznym plikiem jest biblioteka PASLIB (zawiera 16,8 KB kodu i 2,3 KB danych), z której wybiera się tylko te moduły, do których wystąpiły odwołania z programu wynikowego (zwykle od 2 do 6 KB). Inne pliki są dołączane w całości (nie są bibliotekami) w zależności od rodzaju użytych rozszerzeń języka PASCAL MT+.

Program wynikowy składa się z modułów generowanych oddzielnie dla każdej procedury, funkcji i programu głównego w porządku zgodnym z kolejnością ich części instrukcyjnych. Ograniczenie na wielkość modułu dla procedury lub funkcji wynosi 2 KB.

Szybkość kompilacji wynosi średnio 2000 B/min. Przykładowo — kompilacja zestawu modułów źródłowych o wielkości 1 MB wynosiła ok. 8 godz. pod nadzorem systemu CP/M.

Zawartość kodu jest dość dobra, np. dla poniższego przykładu wynosi 887 bajtów (625+116+146) oraz 204 bajty (200+2+2) danych.

Schemat podziału pamięci w czasie wykonywania programów pascalowych (rys. 3) może być elastycznie modyfikowany przez użytkownika za pomocą odpowiednich parametrów kompilacji i programu łączącego.



Rys. 3. Podział pamięci w czasie wykonania programu

Skład, którego początek wskazuje standardowa zmienna SYSTEM, występuje tylko w przypadku użycia zmiennych wskaźnikowych. Podobnie, stos na segmenty danych procedur wywoływanych rekurencyjnie (jego koniec wskazuje zmienna @SFP) występuje tylko wtedy, gdy włączony jest parametr kompilacji S (dozwolone użycie procedur i funkcji rekurencyjnych).

PRZYKŁAD

W celu zilustrowania praktycznego działania w systemie PASCAL MT+ i zaprezentowania możliwości językowych, poniżej przedstawiono przykładowy program LICZ. Celem tego programu jest wczytanie ze standardowego pliku wejściowego (klawiatura) jednoliterowego kodu operacji (W, Z, S) i wykonanie jej na pliku tekstowym LICZ.TST znajdującym się na dyskietce napędu B. Operacje te oznaczają: liczenie wierszy (W), liczenie znaków (Z) oraz zatrzymanie programu (S). Wszystkie informacje program wpisuje na standardowy plik wyjściowy (monitor ekranowy).

```
module LICZ LW;

  var f: external text;          (*obiekt importowany*)

  procedure lw;

  (* licz wiersze na pliku tekstowym f *)

    var l: integer;              (*licznik wierszy*)

  begin

    l := 0;

    while not eof(f) do

    begin

        l := l +1; readln(f)

    end;

    writeln('Liczba wierszy na pliku źródłowym: ', l)

  end;

modend.
```

Wydruk 1. Moduł źródłowy LICZ_LW

```
program LICZ(f,input,output);
(* licz wiersze i znaki na pliku tekstowym f,          *)
(* któremu jest przypisany zbiór danych B:LICZ.TST *)

  var
    f: text;                     (* obiekty *)
    z: char;                     (* globalne *)

  external [1] procedure lw;(*zewnętrzna procedura w nakładce Nr 1
  external [2] procedure lz;(*zewnętrzna procedura w nakładce Nr 2

begin
  writeln('Są dostępne operacje na pliku tekstowym B:LICZ.TST:');
  writeln(' W - liczenie Wierszy,');
  writeln(' Z - liczenie Znaków,');
  writeln(' S - Stop.');
  assign(f,'B:LICZ.TST');
  repeat
    reset(f);
    if ioresult <> 255 then
      begin
        (*otwarcie pliku B:LICZ.TST bez błędów*)
          write('Wprowadź znak: W, Z, S -'); read(z); writeln;
          case z of
            'W', 'w': lw;
            'Z', 'z': lz;
          else
            if not (z in ['S', 's']) then
              writeln('Niedozwolony znak')
          end (*case*)
      end
    else writeln('Błąd przy otwarciu pliku B:LICZ.TST')
  until z in ['S', 's'];
  writeln('Koniec programu.')
end.
```

Wydruk 2. Moduł źródłowy LICZ

Cały program został — w celach dydaktycznych — podzielony na trzy moduły, oddzielnie kompilowane: część

```
module LICZ LZ;

  var f: external text;          (*obiekt importowany*)

  procedure lz;
(*licz znaki na pliku tekstowym f*)

    var l: integer;              (*licznik znaków*)

  begin
    l := 0;
    while not eof(f) do
    begin
      while not eoln(f) do
      begin
        l := l +1; get(f)
      end;
      readln(f)

  end;
  writeln('liczba znaków na pliku źródłowym:', l)
end;

modend.
```

Wydruk 3. Moduł źródłowy LICZ_LZ

rezydentną — program główny (LICZ) i dwie nakładki (LICZ_LW, LICZ_LZ). Zmienna plikowa f (typu text) jest zasobem dzielonym między modułami — zadeklarowana jest globalnie w module LICZ i jest importowana przez obie nakładki, tj. moduły LICZ_LW i LICZ_LZ. Teksty źródłowe poszczególnych modułów podano na wydrukach.

* * *

System SPP jest wygodnym, prostym i efektywnym narzędziem pracy programisty, szczególnie przydatnym do przygotowania, redagowania i kompilowania tekstów w języku PASCAL. Uznając założenia systemu SPP, wypada stwierdzić, że ma on niewiele wad — przypadkowe (zupełnie nieczytelne) skojarzenie kluczy kontrolnych z poleceniami edytora oraz przykre przerywanie pracy edytora (powrót do systemu operacyjnego) przy próbie zapisania redagowanego tekstu bez uprzedniego zezwolenia na zapis na danej dyskietce. Należy zwrócić uwagę na ogromną prostotę i zwartość edytora w systemie SPP (znacznie większą w porównaniu z edytorem WordStar), w pełni wystarczającą do redagowania tekstów programów pascalowych, co stanowi kolejną egzemplifikację znanej dewizy „małe (i wyspecjalizowane) jest piękne", spopularyzowanej w dziedzinie programowania przez autorów systemu UNIX. Załamanie pracy systemu SPP (i strata zawartości buforów) zdarza się raczej rzadko.

Kompilator PASCAL MT+ akceptuje pełny język PASCAL zgodny ze standardem ISO. Oprócz tego, zawiera bogactwo różnorodnych rozszerzeń i dodatkowych narzędzi. Nie wszystkie rozszerzenia są zgodne z duchem, a nawet ze składnią PASCALA. Pozwala to na korzystanie ze wszystkich zasobów mikrokomputera i systemu operacyjnego bezpośrednio w języku PASCAL MT+ (i czyni zbędnym programowanie w języku asemblera), lecz kosztem gwałtownego wzrostu niebezpieczeństwa w programowaniu. Rzeczywiście, zasadne są częste zastrzeżenia, nawet w podręczniku PASCALA MT+, że jest to język tylko dla doświadczonych programistów — jeśli chce się tworzyć programy nie tylko efektywne, ale i niezawodne.

Można oczekiwać dalszego wzrostu popularności systemu SPP i kompilatora PASCAL MT+, a nawet — tendencji do wzorowania się na nim przez inne firmy mikrokomputerowe.

Obecnie oprogramowanie systemowe (systemy operacyjne CP/M i MP/M, PASCAL MT+) jest wystarczające do wykonywania programów, ale nie do pracy nad ich pełnym przygotowaniem (projektowaniem i uruchamianiem). Szczególną uwagę zwraca niebezpieczeństwo używania większości rozszerzeń języka. Pożądany byłby — moim zdaniem — system o prostocie i efektywności podobnej do systemu SPP, wspomagający programistę na etapie projektowania oraz testowania i uruchamiania programów.

LITERATURA

[1] Addymann A. M.: Specification for the Computer Programming Language Pascal. Second draft proposal, Document ISO/TC 97/SC5 N595, 1981

[2] DIGITAL RESEARCH: Speed Programming Package — User's Guide, Release 5.2. Pacific Grove, California, USA, 1982

[3] MT MicroSystems Inc.: Pascal MT+ — User's Guide, Release 5, Lifeboat Associates, New York, 1981

[4] Pournelle J.: The Debate Goes on — A Discussion of the Future of Microcomputer Languages. Byte, pp. 312—328, August 1983

WIESŁAW WILCZYŃSKI
Warszawa

PLZ — język programowania mikroprocesorów (2)

W drugiej części opisu języka PLZ omówię ważniejsze instrukcje, strukturę programu oraz zasady łączenia modułów.

INSTRUKCJE

Zwykła instrukcja IF umożliwia wykonanie odpowiedniego fragmentu programu w zależności od wartości logicznej sformułowanego warunku, np.:

```
IF DAY>LAST_DAY THEN LAST_DAY:=DAY FI
```

Instrukcja jest zamknięta słowami IF i FI, zaś warunek logiczny może przybierać postać różnorakich wyrażeń arytmetyczno-logicznych. Inna postać zwykłej instrukcji IF umożliwia wybór (w zależności od warunku logicznego) i wykonanie jednego z dwóch fragmentów programu, np.:

```
IF DAY>LAST_DAY THEN LAST_DAY:=DAY ELSE LAST_
_DAY:=DAY-1 FI
```

Pomiędzy słowami THEN i ELSE oraz ELSE i FI można umieszczać dowolny ciąg instrukcji, a w szczególności kolejne instrukcje IF. Taka wewnętrzna instrukcja IF nazywa się zagnieżdżoną. W przypadku dużej liczby poziomów zagnieżdżenia, zapis staje się nieczytelny. W celu poprawienia czytelności warunków można stosować złożoną instrukcję IF, tzn. zamiast konstrukcji:

```
IF (warunek_1) THEN
   IF (warunek_2) THEN
         (zdanie)
   FI
FI
```

stosuje się następującą:

```
IF (warunek_1) ANDIF (warunek_2) THEN
   (zdanie)
FI
```

W języku PLZ istnieją dwa specjalne operatory warunku ANDIF i ORIF.

Gdy zbiór działań jest selekcjonowany według wartości pewnego obiektu typu dyskretnego, wówczas dla zwiększenia efektywności i czytelności można posłużyć się instrukcją CASE, zamiast — złożoną instrukcją warunkową, np.:

```
IF LAST_CHAR
   CASE 'S' THEN STRING += 1
   CASE 'A' THEN STRING -= 2
   CASE 'D' THEN STRING += 3
   ELSE
         STRING:=NULL
FI
```

Nazwa LAST_CHAR odgrywa rolę wyróżnika, zaś nazwa wymieniona po słowie CASE — rolę etykiety wariantu. Człon ELSE jest opcjonalny i wykonuje się go wtedy, gdy wyróżnik nie jest zgodny z żadną z wymienionych etykiet. W podanym przykładzie użyto instrukcji przypisania w każdym wariancie, ale w ogólnym przypadku może to być dowolne zdanie języka PLZ.

Dr inż. WIESŁAW WILCZYŃSKI ukończył studia na Wydziale Elektroniki Politechniki Warszawskiej. Od 1971 roku pracuje w Instytucie Telekomunikacji PW. Zajmuje się zagadnieniami związanymi z programowanym sterowaniem central telefonicznych.

Instrukcja DO umożliwia kontrolowane powtarzanie wykonywania segmentu programu. W celu kontroli działania instrukcji DO należy określić, który segment programu ma być powtarzany i ile razy należy powtórzyć wykonanie. Do określenia segmentu, który należy wykonywać służy para nawiasów DO i OD, np.:

```
DO
   zdanie_1
   zdanie_2
   ...
OD
```

Tak zapisana instrukcja pętli jest pętlą nieskończoną, o ile żadne ze zdań tworzących ciało pętli nie powoduje jej opuszczenia. W języku PLZ nie ma części sterującej instrukcji DO i określenie liczby powtórzeń należy do programisty. Opuszczenie pętli umożliwiają instrukcje EXIT lub RETURN, które mogą stanowić człon instrukcji IF, np.:

```
X:=0
Y:=0
DO
   X+=1
   IF X>MAX_NUMBER THEN EXIT FI
   Y+=1
OD
```

Postać warunku opuszczenia pętli oraz jego położenie w ciele pętli jest całkowicie dowolne. Po wykonaniu instrukcji EXIT, sterowanie jest przenoszone bezpośrednio za nawias zamykający OD. Jest oczywiste, że pętla może zawierać kilka instrukcji EXIT lub RETURN związanych z odpowiednimi warunkami. Instrukcją o przeciwnym działaniu jest REPEAT, która umożliwia powtórzenie wykonania części pętli, zawartej między słowami DO i REPEAT. Podobnie jak instrukcji EXIT, instrukcji REPEAT można używać łącznie z instrukcją IF, np.:

```
DO
zdanie_1
   IF warunek_1 THEN REPEAT FI
   zdanie_2
   IF warunek_2 THEN EXIT FI
OD
```

Jeśli zostanie spełniony warunek_1, to nastąpi powtórne wykonanie zdania_1. O ile zostanie spełniony warunek_2, to nastąpi opuszczenie pętli, a w przeciwnym przypadku — powrót do wykonania zdania_2.

```
1.  INTERNAL
2.       TEST ARRAY (5,7,8 WORD)
3.       HELP WORD
         ...
4.       I:=0, J:=0, K:=0
5.  ROW_LOOP:   I:=0
6.              DO
7.                 J:=0
8.  COLUMN_LOOP:   DO
9.                    K:=0
10. DEEP_LOOP:        DO
11.                      IF K>7 THEN EXIT FI
12.                      HELP := TEST (I,J,K)
13.                      K+=1
14.                      IF HELP=END THEN EXIT FROM ROW_LOOP FI
15.                   OD
16.                   J+=1
17.                   IF I>6 THEN EXIT FI
18.                OD
19.                I+=1
20.                IF I>4 THEN EXIT FI
21.             OD
```

Przykład 1

Podobnie jak w innych językach, w PLZ dopuszcza się pętle zagnieżdżone, bez ograniczeń co do głębokości zagnieżdżeń. Zagnieżdżone instrukcje DO są bardzo pomocne przy przeglądaniu tablic wielowymiarowych. Konstrukcja przedstawiona w przykładzie 1 przypisuje zmiennej HELP

kolejne elementy tablicy TEST. Jeżeli w tablicy TEST poszukujemy elementu o wartości END i po jego znalezieniu istnieje potrzeba przekazania sterowania do linii 22, umożliwia to wielopoziomowa instrukcja EXIT, występująca w linii 14. Człon EXIT FROM ROW_LOOP powoduje przekazanie sterowania do następnej instrukcji po nawiasie zamykającym OD, którego nawias otwierający DO poprzedzony jest etykietą wymienioną w instrukcji EXIT FROM. Symetrycznie do instrukcji EXIT FROM działa instrukcja REPEAT FROM, powodująca powtórzenie działań zagnieżdżonych w pętli od wskazanej etykiety, np.:

```
TEST_LOOP:DO
            ...
            DO
            IF warunek THEN REPEAT FROM TEST_LOOP FI
            OD
            ...
          OD
```

STRUKTURA PROGRAMU

Reguły konstruowania programu w języku PLZ pozwalają pogrupować w zadanym porządku opisy zmiennych oraz działań z nimi związanych, a zatem zdekomponować program na mniejsze segmenty. W konsekwencji uzyskuje się:
— podział oprogramowania na mniejsze segmenty (moduły), które mogą być oddzielnie kompilowane i uruchamiane, co ułatwia projektowanie, rozbudowę i pielęgnację oraz pozwala zwiększyć czytelność programu
— odizolowanie wzajemnie niezależnych segmentów programu oraz zwiększenie kontroli powiązań pomiędzy współpracującymi segmentami, a co za tym idzie — eliminację niepożądanych efektów ubocznych.

Moduły i zasięg nazw

Elementami języka umożliwiającymi strukturalizację programu są moduły. Program napisany w PLZ składa się z jednego lub kilku modułów, z których każdy ma następującą konstrukcję nawiasową:

```
Nazwa_modułu MODULE
! Treść modułu!
END Nazwa_modułu.
```

Moduły nie mogą się zagnieżdżać. Na treść modułów składają się opisy obiektów oraz czynności, które będą wykonywane na tych obiektach.

W programie składającym się z kilku modułów, programista określa, które segmenty programu mają dostęp przez nazwę do określonych zmiennych lub procedur. Segment programu, z którego może być dokonane odwołanie, określa zasięg nazwy (ang. scope). Następujące słowa: GLOBAL, EXTERNAL, INTERNAL, LOCAL, stowarzyszone z nazwą, determinują jej zasięg.

Wszystkie nazwy zadeklarowane jako INTERNAL mają zasięg ograniczony tylko do modułu. Postać deklaracji INTERNAL jest następująca:

```
INTERNAL <lista deklaracji zmiennych lub procedur>
```

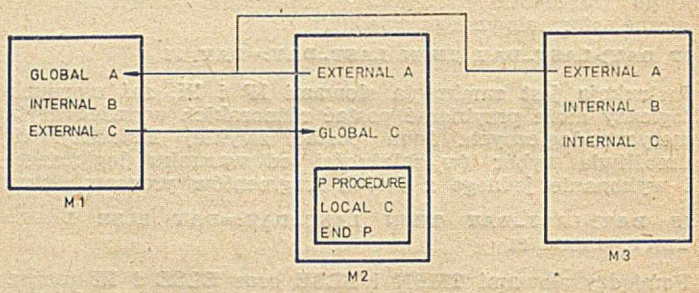
W przypadku gdy określone obiekty mają być dostępne nie tylko w jednym module, stosuje się następujące reguły:
- w jednym z modułów nazwa zmiennej lub procedury deklarowana jest jako GLOBAL
- w pozostałych modułach, w których ma być dostępna, ta sama nazwa deklarowana jest jako EXTERNAL.

Widoczność nazw stowarzyszonych ze słowem LOCAL jest ograniczona do procedury, w której występuje. Jest to jedyna możliwa deklaracja w treści procedury i może występować tylko w procedurze.

```
1.  LEGAL_CONFLICT MODULE
2.  TYPE  IREC RECORD (B BYTE, C BYTE)
3.        JREC RECORD (A BYTE, B BYTE)
4.  INTERNAL
5.        A,B,C BYTE
6.        I IREC
7.        J JREC
8.  GLOBAL
9.        P PROCEDURE(J BYTE) RETURNS (K BYTE)
10.         LOCAL A,B BYTE
11.         ENTRY
12.         B := I.B
13.         A := J
14.         I.C := C
15        END P
16. END LEGAL_CONFLICT
```

Przykład 2

Znaczenie słów zasięgu wyjaśniono w przykładzie 2. W linii 1 zadeklarowano nazwę LEGAL_CONFLICT jako nazwę modułu. W liniach 2 i 3 zdefiniowano typy IREC i JREC, które są rekordami o dwóch polach typu BYTE. W liniach 4—7 przesłonięto deklarację zmiennych A, B, C typu BYTE, zmiennej I typu IREC oraz zmiennej J typu JREC. Obiekty te nie są widoczne na zewnątrz modułu ze względu na użycie słowa zasięgu INTERNAL. W liniach 9—15 zdefiniowano procedurę P, która może być wywoływana z innych modułów (GLOBAL w linii 8) z formalnym parametrem wejściowym J typu BYTE i z formalnym parametrem wyjściowym K typu BYTE. W linii 10 zadeklarowano obiekty A i B typu BYTE niewidoczne na zewnątrz procedury P. W linii 12, lokalnej zmiennej B typu BYTE przypisywana jest wartość pola B zmiennej I typu IREC, zaś w następnej linii lokalna zmienna A przybiera wartość wejściowego parametru J. W linii 14, pole C zmiennej I typu IREC otrzymuje wartość zmiennej C zadeklarowanej w linii 5.



Rys. 1. Ilustracja zasięgu zmiennych

Rozpatrzmy program składający się z trzech modułów M1, M2, M3, przedstawionych na rysunku 1. Zmienne A zadeklarowane w modułach M1, M2, M3 są reprezentowane przez ten sam adres, zaś zmienne B w modułach M1 i M3 — przez dwa różne adresy w pamięci. Zmienne C w modułach M1 i M2 są różne od zmiennej w module M3, zaś lokalna zmienna C w globalnej procedurze P ma dynamicznie przydzielany adres.

Zasięg nazwy typu definiowanego przez użytkownika oraz nazwy stałej jest zawsze ograniczony do modułu. Typ zdefiniowany przez użytkownika w jednym module i powtórzony w innym ma dokładnie to samo znaczenie w obu modułach, podobnie jak definicja CONSTANT. W celu zautomatyzowania procesu kopiowania deklaracji EXTERNAL, TYPE i CONSTANT możliwe jest użycie — w ciele modułu — polecenia:

```
$ INCLUDE "nazwa modułu"
```

które w trakcie kompilacji programu spowoduje wykonanie w tym module kopii deklaracji i definicji obiektów wskazanych przez nazwę pola.

Procedury

Możliwość definiowania i wywoływania procedur jest w PLZ — podobnie jak w innych językach — jednym z podstawowych narzędzi programowania. Procedura stanowi środek dekompozycji dużych zadań na mniejsze, pozwala zmniejszyć rozmiary programu oraz rozszerzyć język o nowe operacje.

Procedura w PLZ określa ciąg działań (zazwyczaj sparametryzowany) identyfikowany jednoznacznie przez nazwę i posiadający ustalony sposób wymiany informacji z pozostałymi częściami programu. Procedury opisuje się za pomocą definicji, zaś ich wykonanie jest inicjowane przez instrukcję wywołania. Po wykonaniu działań wynikających z treści procedury, wykonanie programu jest kontynuowane od instrukcji znajdującej się bezpośrednio za wywołaniem procedury.

Definicja procedury składa się z nazwy, nagłówka oraz ciała procedury zaczynającego się słowem ENTRY i kończącego się słowem END oraz nazwą procedury. W zależności od słowa zasięgu poprzedzającego nazwę, procedura może mieć zasięg modułowy lub wielomodułowy. Nagłówek rozpoczyna się od słowa PROCEDURE, po którym następuje lista formalnych parametrów wejściowych oraz lista parametrów wyjściowych, poprzedzonych słowem RETURNS. Parametrami mogą być obiekty typu prostego i typu zdefiniowanego przez użytkownika. Kolejnym ele-

mentem nagłówka (opcjonalnym) jest deklaracja zmiennych lokalnych.

Procedura musi być zadeklarowana, zanim nastąpi odwołanie do niej. Parametry procedury są zmiennymi, ale mają pewne cechy odróżniające je od zmiennych statycznych, a mianowicie:

- mogą mieć te same nazwy co zmienne zadeklarowane na zewnątrz procedury
- ich widoczność rozciąga się tylko na obszar procedury, a przydział pamięci dla nich jest dynamiczny
- parametry wejściowe są automatycznie inicjowane w chwili wywołania procedury, zaś wartość parametrów wyjściowych jest im nadawana w chwili wykonania instrukcji powrotu z procedury
- parametry wejściowo-wyjściowe mogą być używane jako zmienne lokalne.

Procedura może zawierać deklaracje zmiennych lokalnych, które podobnie jak parametry są dynamiczne i przestają istnieć po wykonaniu instrukcji powrotu. Zmienne lokalne różnią się pod pewnymi względami od parametrów procedury, a mianowicie:
— nie są bezpośrednio związane z wywołaniem i powrotem z procedury
— mogą być typu ARRAY lub RECORD
— w chwili wywołania procedury ich wartości nie są określone
— w deklaracji nie można nadać im wartości początkowych (jest to możliwe tylko przy użyciu instrukcji przypisania w ciele procedury).

Rozpatrzmy przykład procedury zamieniającej miejscami dwa bajty:

```
SWAP_BYTES PROCEDURE (A,B BYTE)
           RETURNS   (C,D BYTE)
  ENTRY
  C:=B
  D:=A
END SWAP_BYTES
```

Pierwsze dwie linie zawierają nazwę procedury SWAP_-BYTE oraz nagłówek wraz z deklaracjami parametrów wejściowych i wyjściowych. Ciało procedury tworzą dwie instrukcje przypisania. Wywołanie procedury w PLZ ma postać nazwy procedury, po której następuje lista parametrów formalnych ujęta w nawiasy. Wywołanie przedstawionej procedury ma postać:

```
X,Y:=SWAP_BYTE(X,Y)
```

Ta specjalna postać instrukcji przypisana jest przeznaczona dla procedur mających więcej niż jeden parametr wyjściowy.

Według reguł programowania zstępującego, z procedury można wywoływać inne procedury. Taka konstrukcja programowa jest możliwa w PLZ, przy czym należy pamiętać, że:

- zmienne lokalne zadeklarowane w procedurze są dynamiczne i przestają istnieć w chwili opuszczania procedury
- wszystkie obiekty zadeklarowane na zewnątrz procedury wywoływanej, a więc także w procedurze wywołującej, są dostępne wewnątrz procedury wywoływanej, wskutek czego mogą powstać efekty uboczne, o ile nie zastosowano przesłaniania nazw
- jeżeli w otoczeniu procedury istnieje zmienna, której nazwa pokrywa się z nazwą parametru lub zmiennej lokalnej, to jest ona niedostępna wewnątrz wywołanej procedury (efekt przesłaniania)
- wszystkie procedury w PLZ mogą wywoływać same siebie (są rekurencyjne).

Jeżeli skutkiem wykonania procedury jest wyznaczenie wartości, to możliwe jest przekazanie tej wartości przez nazwę procedury. Procedura ma wówczas sens funkcji i może być używana w wyrażeniach. W procedurach funkcyjnych istotne jest unikanie efektów ubocznych. Procedury takie deklaruje się dokładnie tak, jak zwykłe procedury — pod warunkiem, że lista parametrów wyjściowych zawiera dokładnie jeden parametr (p. przykład 3). Wywołanie przedstawionej funkcji, obliczającej średnią M elementów tablicy B, ma następującą postać:

```
AVG:= AVERAGE (#B[0], M)
```

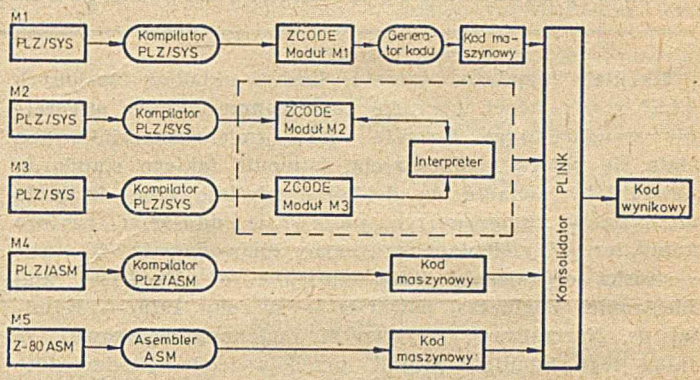
Instrukcja RETURN jest zawsze umieszczana niejawnie na końcu procedury. Gdy wykonanie procedury dobiegnie do słowa END, sterowanie przebiegiem programu powraca do miejsca wywołania. Powrót może nastąpić również przed osiągnięciem słowa END, jeżeli użyta zostanie jawna instrukcja RETURN lub EXIT (np. w połączeniu z instrukcją IF).

```
1.  AVERAGE PROCEDURE(PX ¬WORD, N WORD) RETURNS (A WORD)
2.      LOCAL SUM WORD, I BYTE
3.      ENTRY
4.      SUM:=0, I:=1
5.      DO
6.          IF I>N THEN EXIT FI
7.          SUM+=PX¬
8.          PX := INC PX, I+=1
9.      OD
10.     A := SUM/N
11. END AVERAGE
```

Przykład 3

ŁĄCZENIE MODUŁÓW

Projektanci języka PLZ (dokładniej PLZ/SYS) brali pod uwagę całą rodzinę zgodnych języków PLZ — tak, aby było możliwe łączenie modułów pisanych w różnych językach. Jeden z nich, PLZ/ASM, pozwala pisać bardzo efektywne programy na poziomie asemblera. Możliwe jest też pisanie modułów w standardowym języku asemblera firmy ZILOG. Wymienione możliwości przedstawiono na rysunku 2. W celu ich zilustrowania rozważymy prosty przykład.



Rys. 2. Łączenie modułów napisanych w różnych językach

Załóżmy, że do dyspozycji mamy nieprzekraczalną wielkość 10 000 bajtów pamięci. Niech program składa się z trzech modułów o nazwach: SCAN, LOOKUP i MAIN. Wskutek kompilacji i użycia generatora kodu otrzymano moduły o następujących parametrach (kod maszynowy):

Nazwa	Wielkość (bajty)	Czas wykonania (s)
SCAN	2000	15
LOOKUP	2000	12
MAIN	8000	3

Sumaryczna objętość kodu wynosi 12 000 bajtów, co przekracza założoną wielkość. Rozważmy drugą możliwość, kiedy wszystkie moduły są w postaci kodu pośredniego (ZCODE) i dołączony jest moduł interpretera.

Nazwa	Wielkość (bajty)	Czas wykonania (s)
SCAN	1000	60
LOOKUP	1000	50
MAIN	4000	10
interpreter	2000	—

Sumaryczna objętość programu (łącznie z interpreterem) wynosi 8000 bajtów, ale czas wykonania znacznie się wydłuża. Rozważmy trzecią możliwość:

Nazwa	Wielkość (bajty)	Czas wykonania (s)
SCAN (kod maszynowy)	2000	15
LOOKUP (kod maszynowy)	2000	12
MAIN (ZCODE)	4000	10
interpreter	2000	—

Ta hybrydowa konstrukcja wydaje się być optymalna, gdyż jak wykazuje praktyka, w systemach sterujących zwykle wykonanie 10% całego programu pochłania 90% czasu procesora.

LITERATURA

[1] Bass C.: PLZ — A Family of System Programming Languages for Microprocessors. Computer, Vol. 11, No. 3, pp. 34—39, 1978
[2] Conway R., Gries D.: Introduction to Microprocessor Programming Using PLZ. Wintrop Publishers Inc., 1979
[3] ZILOG Inc.: PLZ Version 2.0 User's Guide. Order No 03-0094-00, Cupertino (CA), September 1978.

ZBIGNIEW BANASIK
Warszawa

Żyrardowskie Zakłady Tkanin Technicznych
96-300 ŻYRARDÓW, ul. Okrzei 51
zakupią
stację dyskową MERA 9425 lub MERA 9450
Oferty przyjmuje:
Dział Informatyki, tel. 20-31 do 35, w. 284
EO/141/K/85

Projekt normy IEEE — arytmetyka zmiennoprzecinkowa

Wskutek ciągłego wzrostu liczby układów scalonych VLSI realizujących operacje zmiennoprzecinkowe, stworzenie standardowego formatu liczb zmiennoprzecinkowych stało się palącą koniecznością. Istnienie takiego standardu pozwoliłoby ponadto na dokładne zdefiniowanie działania algorytmów numerycznych, niezależnie od typu użytego komputera. Wymienione przyczyny spowodowały, że amerykańska organizacja IEEE (Institute of Electrical and Electronics Engineers) powołała — w roku 1970 — Komitet ds. Normalizacji Arytmetyki Zmiennoprzecinkowej. W skład tego komitetu weszli przedstawiciele producentów oraz użytkowników maszyn cyfrowych. Ostateczna wersja omawianego dokumentu uzyskała akceptację większości członków (przeciwni byli przedstawiciele producentów!) i została przedstawiona do zatwierdzenia przez IEEE [1].

Opracowany dokument jest dość obszerny i zawiera m.in.:

— definicje terminów używanych w treści (p. str. 30 bieżącego numeru)

— opis reprezentacji (formatów) liczb zmiennoprzecinkowych

— opis sposobów zaokrąglania wyników

— opis dozwolonych działań (operacji)

— opis tzw. nie-liczb (ang. Not a Number — NaN), czyli takich kombinacji bitów, które nie mogą być traktowane jako liczby

— opis sposobu normalizowania liczb

— wykaz błędów, które mogą pojawić się w czasie obliczeń oraz opis sposobu ich sygnalizowania użytkownikowi. Ponadto, zdefiniowano zalecane funkcje i predykaty.

Powszechnie przyjętym sposobem zapisu liczby zmiennoprzecinkowej jest przedstawienie jej jako pary (wykładnik, mantysa). Mantysa jest liczbą z przedziału [1/2, 1), choć niekiedy stosuje się inną normalizację. Wykładnik jest potęgą, do której należy podnieść 2, aby po pomnożeniu przez mantysę otrzymać żądaną liczbę. Najbardziej znaczący bit unormowanej mantysy jest znany i zawsze równy jedności, można go zatem pominąć w zapisie. Do pełnego zapisu liczby potrzebny jest także bit znaku.

Zapis liczby zmiennoprzecinkowej w postaci trójki:
$v = (s, e, f)$

gdzie f jest znormalizowaną mantysą, nazywa się postacią znormalizowaną liczby v. Liczba ta ma wartość:

$v = (-1)^s 2^e (1.f)$

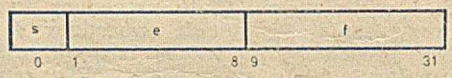
Będziemy używać również pojęcia liczby nieunormowanej, tzn. takiej, której mantysa ma pierwszy bit równy zeru oraz:

$v = (-1)^s 2^e (0.f)$

Obydwa pojęcia odgrywają w normie IEEE podstawową rolę.

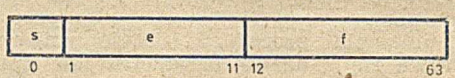
FORMATY

Norma IEEE dopuszcza cztery podstawowe formaty zapisu liczb zmiennoprzecinkowych: dwa podstawowe (ang. basic) i dwa rozszerzone (ang. extended). Formaty podstawowe pojedynczej i podwójnej precyzji są formatami dla wyników końcowych, natomiast rozszerzone — dla wyników pośrednich.



Rys. 1. Format podstawowy pojedynczej precyzji (s — bit znaku, e — wykładnik, f — mantysa)

Format podstawowy pojedynczej precyzji przedstawiono na rysunku 1. Liczba zajmuje 32 bity według następującego podziału:
s — bit 0, znak liczby (0 — plus, 1 — minus)
e — bity 1—8, wykładnik
f — bity 9—31, mantysa.
Wartość liczby:
— jeśli $e=255$ i $f\neq 0$, to $v=NaN$ (nie liczba)
— jeśli $e=255$ i $f=0$, to $v=(-1)^s\cdot\infty$
— jeśli $0<e<255$, to $v=(-1)^s 2^{e-127}(1.f)$
— jeśli $e=0$ i $f\neq 0$, to $v=(-1)^s 2^{-126}(0.f)$
— jeśli $e=0$ i $f=0$, to $v=(-1)^s 0$ (czyli zero).



Rys. 2. Format podstawowy podwójnej precyzji (oznaczenia jak na rys. 1)

Format podstawowy podwójnej precyzji przedstawiono na rysunku 2. Liczba zajmuje 64 bity według następującego podziału:

s — bit 0, znak liczby
e — bity 1—11, wykładnik
f — bity 12—63, mantysa.

Wartość liczby:

— jeśli $e=2047$ i $f\neq 0$, to $v=NaN$
— jeśli $e=2047$ i $f=0$, to $v=(-1)^s\cdot\infty$
— jeśli $0<e<2047$, to $v=(-1)^s 2^{e-1022}(1.f)$
— jeśli $e=0$ i $f\neq 0$, to $v=(-1)^s 2^{-1022}(0.f)$
— jeśli $e=0$ i $f=0$, to $v=(-1)^s 0$ (czyli zero).

dokończenie na str. 21

Jedną z bardziej spektakularnych cech najnowszych komputerów osobistych są ich możliwości graficzne. Otwierają się nowe perspektywy nie tylko przed fanatykami gier komputerowych, ale i przed projektantami najrozmaitszych urządzeń. Do poważniejszych zastosowań niezbędne są jednak odpowiednie urządzenia we-wy i oprogramowanie. Spadające ceny sprzętu świadczą, że grafika komputerowa niebawem znajdzie się w zasięgu możliwości co bardziej zdeterminowanych amatorów mikroinformatyki. Dlatego z pewnym wyprzedzeniem spróbujemy przekazać Czytelnikom pakiet odpowiednich informacji. Rozpoczynamy od skrótowego opisu normy GKS, który przyda się do zrozumienia następnych tekstów na ten temat.

Jako ciekawostkę podajemy, że ilustracje do poniższego tekstu zostały wykonane za pomocą plotera podłączonego do ELWRO-513 (również opisy na rysunkach).

Norma GKS

Trudno uwierzyć, że grafika komputerowa ma już ponad trzydzieści lat! Pierwsze monitory graficzne pojawiły się w 1951 roku, natomiast pisaki X-Y — w 1953. Graficzne urządzenia wejściowe stosowane są w świecie od roku 1958 (pióro świetlne). Początek kolorowych monitorów graficznych datuje się na lata 1962—65.

Do niedawna cena zestawu podstawowych graficznych urządzeń zewnętrznych była bardzo wysoka (ok. 400 tys. dol.). Obecnie kwota ta spadła do tego stopnia (kilkadziesiąt razy), że mówi się już o grafice mikrokomputerowej.

W latach siedemdziesiątych prowadzone były intensywne prace nad standardem dla grafiki komputerowej (m.in. GSPC Core w USA, GINO-F w Wielkiej Brytanii, IDIGS w Holandii i Norwegii, GKS w RFN). Dopiero w 1982 roku ISO (International Organization for Standardization) uznała **GKS** (Graphical Kernel System — system jądra graficznego) za normę międzynarodową.

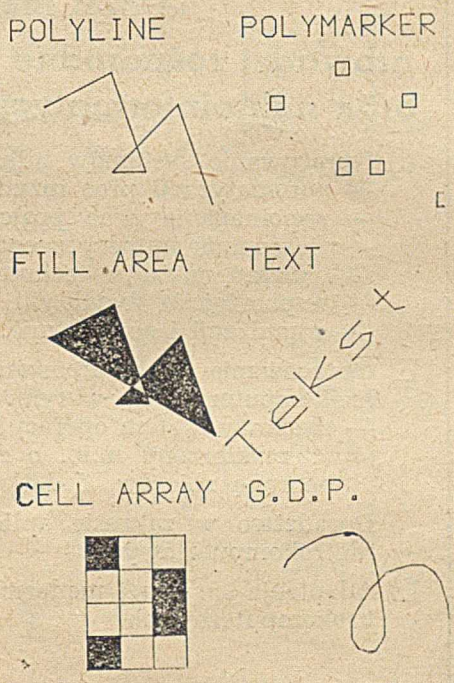
Obecnie wszystkie liczące się w świecie firmy komputerowe stosują się do tej normy, ponieważ zgodność z GKS jest niezbędna dla sukcesu w sprzedaży sprzętu i oprogramowania graficznego.

Szczegółowy opis normy znaleźć można w [1]. Informacje na temat powstania GKS zaczerpnąć można z [2]. Godną polecenia jest też książka [3].

Rys. 1. Miejsce GKS w programie użytkowym

Do graficznych urządzeń wyjścia zaliczone są: monitory graficzne monochromatyczne i kolorowe, pisaki X-Y oraz drukarki graficzne. Przedmiotem normy GKS są zasady tworzenia i przetwarzania obrazów. Dla GKS obrazy składają się ze skończonej ilości podstawowych bloków, tzw. prymitywów, do których zaliczono:

— łamaną (ang. **POLYLINE**), czyli ciąg odcinków o wspólnych końcach
— zbiór punktów (ang. **POLYMARKER**) znaczonych pewnymi symbolami graficznymi
— obszar ograniczony wielokątem (ang. **FILL AREA**), zakreskowany według pewnej zasady lub wypełniony ustalonym kolorem
— tekst alfanumeryczny (ang. **TEXT**) o ustalonych parametrach
— obraz dwuwymiarowej macierzy (ang. **CELL ARRAY**), której elementami są kody kolorów punktów



Rys. 2. Prymitywy reprezentacji GKS

— łuki krzywych gładkich (ang. **GENERALIZED DRAWING PRIMITIVE**), takich jak okrąg, elipsa czy parabola.

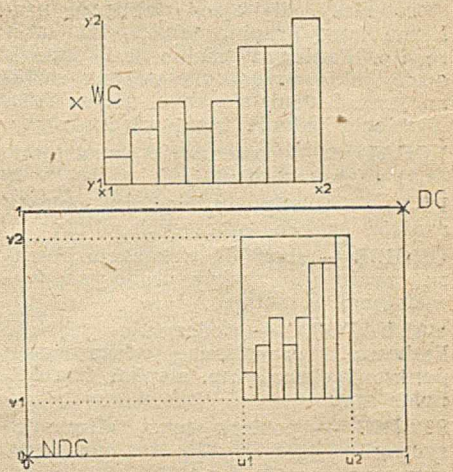
Z każdym prymitywem skojarzone są pewne parametry (atrybuty) określające np.:

— rodzaj linii (ciągła, przerywana, kropkowana lub inna)
— znak graficzny do obrazowania punktów (kropka, kółko, krzyżyk, gwiazdka lub inny)
— sposób wypełniania obszarów (nachylenie i odstęp kresek, kolor lub inny)
— wysokość liter, kierunek pisania tekstu, punkt odniesienia, krój liter, alfabet, itp.
— kolor linii, znaków, obszarów, liter oraz tła.

GKS wyodrębnia trzy układy współrzędnych:

— układ współrzędnych rysunku — WC (ang. world coordinates)
— znormalizowany układ współrzędnych urządzenia — NDC (ang. normalized device coordinates)
— układ urządzenia graficznego — DC (ang. device coordinates).

Znaczenie poszczególnych układów współrzędnych ilustruje rysunek 3.

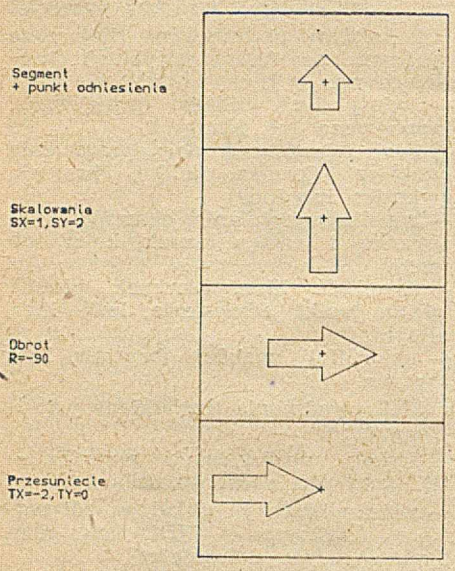


Rys. 3. Układy współrzędnych

Przyjmijmy, że obraz jest pewnym zdaniem sformułowanym w języku o gramatyce atrybutowej. Segmentami nazwiemy „wyrazy", które występują w tym zdaniu. Poprawne wyodrębnienie segmentów ułatwia przetwarzanie obrazu (animację), czyli zmianę atrybutów przyporządkowanych segmentom. Do atrybutów należą:

— punkt odniesienia segmentu
— wektor translacji
— kąt obrotu wokół punktu odniesienia
— skala jednokładności o środku w punkcie odniesienia
— znacznik „obcinania" (ang. clipping)

— znacznik widzialności (znikanie i pojawianie się)
— wskaźnik priorytetu (przesłanianie segmentów o niższym priorytecie)
— znacznik intensywnego świecenia (ang. highlighting).



Rys. 4. Transformacje segmentów

Wśród graficznych urządzeń wejścia należy wymienić: pióro świetlne, monitor „dotykowy", myszkę, drążek sterowniczy, digitajzer, tabliczkę graficzną (ang. graphic tablet), klawiaturę itp. Ponieważ trudno jest przewidzieć, jakie urządzenia wejścia zostaną jeszcze skonstruowane, norma GKS wymienia jedynie typy urządzeń:

LOCATOR — przekazuje współrzędne (x,y) punktu obrazu

PICK — służy do wyboru obiektu spośród innych tworzących obraz

CHOISE — służy do wskazania obszaru obrazu (np. wybór propozycji wymienionej w menu)

VALUATOR — odczytuje wartość liczbową przypisaną wskazanemu punktowi obrazu (np. stopień szarości, kolor itp.)

STRING — pobiera tekst alfanumeryczny

STROKE — przesyła ciąg punktów wyznaczających kontur.

Oprócz podziału urządzeń na typy, GKS uwzględnia trzy rodzaje interakcji:

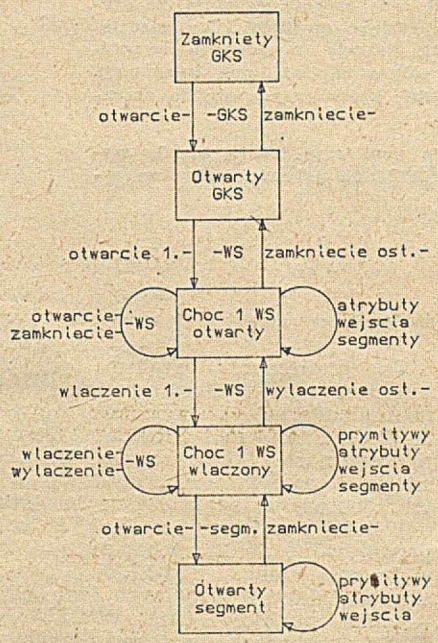
REQUEST — urządzenie wejścia odpowiada na żądanie nadawania, wysłane z programu użytkowego

SAMPLE — program użytkowy i proces wejścia przebiegają równolegle, przy czym rolę nadrzędną pełni program użytkowy

EVENT — program użytkowy i proces wejścia przebiegają równolegle, przy czym rolę nadrzędną pełni proces wejścia.

Dla uniknięcia powtarzania długotrwałego procesu tworzenia obrazu, gotowe rysunki (lub ich fragmenty) można przechowywać w pamięci masowej. Norma GKS nie narzuca sposobu kodowania obrazu, określa jedynie czynności zapisywania i odczytywania.

Do programowania tych czynności przewidziane są funkcje:
— zapisu obrazu w zbiorze; parametrami tej funkcji są: numer logiczny urządzenia graficznego, sposób kodowania obrazu w zbiorze oraz długość i numer rekordu
— rozpoznawania typu danych przed odczytem
— odczyt danych graficznych ze zbioru pamięci masowej
— interpretacja (rozkodowanie) danych i generacja obrazu.



Rys. 5. Graf stanów GKS

Obsługa urządzeń graficznych przebiega według schematu przedstawionego na rysunku 5. Jakiekolwiek odstępstwo od tego schematu lub inne błędy semantyczne sygnalizowane są przez diagnostykę błędów. Komunikat o błędzie zapisywany jest w zbiorze pamięci masowej o nazwie zadeklarowanej w trakcie otwierania GKS.

ANTONI URBAN
Computer Studio Kajkowski

LITERATURA

[1] DIN-66252: Informationsverarbeitung-Graphisches Kernsystem (GKS). Funktionale Beschreibung, Alleinverkauf der Normen durch Beuth Verlag GmbH, Berlin, 1983
[2] Encarnaçao J., Strasser W.: Geräteunabhängige graphische System. R. Oldenbourg Verlag München, Wien, 1981
[3] Hopgood F. R. A., Duce D. A., Gallop J. R., Sutcliffe D. C.: Introduction to the Graphical Kernal System (GKS). Academic Press, London, 1983.

Wojewódzki Klub Techniki i Racjonalizacji
Zakład Usług Technicznych
20-919 Lublin, ul. Marii Curie-Skłodowskiej 3
tel. 213-39, 284-64, 55-80-01 w. 226

prowadzi różnorodne usługi techniczne oraz mikroinformatyczne:

☐ Opracowanie systemów mikroinformatycznych na potrzeby:
— automatyzacji prac inżynierskich
— wspomagania prac projektowych i konstrukcyjnych
— sterowania numerycznego
— obliczeń naukowo-technicznych
— wspomagania dydaktyki
— organizacji i zarządzania.

☐ Opracowanie oprogramowania podstawowego systemów mikrokomputerowych, w tym:
— dyskowy system operacyjny SOMIK kompatibilny z CP/M 2.2,rozszerzony m.in. o zegar, znakowanie zbiorów, datę, ochronę zbiorów, dodatkowe możliwości pracy wsadowej.

☐ Doradztwo w zakresie aplikacji i wykorzystania systemów mikrokomputerowych.

☐ Adaptacja sprzętu i modernizacja architektury systemów mikrokomputerowych.

EO/140/K/85

Pomysł przystawki pozwalającej wykorzystać ZX81 jako woltomierz lub generator funkcji wydaje się być wyjątkowo trafny. Oczywiście, nie należy stawiać sprawy na głowie i kupować ZX81 zamiast woltomierza cyfrowego... Jeżeli jednak znalazł się on już w laboratorium — opisywane rozszerzenie jego możliwości może okazać się użyteczne. Pomysł jest dokładnie na miarę możliwości ZX81 (ideę można łatwo przenieść na inny mikrokomputer, np. ZX SPECTRUM) i to go odróżnia od niektórych szaleńczych aplikacji, jak choćby zastosowanie w intensywnym nadzorze medycznym, od przebiegu którego może zależeć życie pacjenta.

Przystawka do ZX81

Opisana przystawka pozwala na wykorzystanie ZX81 jako generatora funkcji, miernika częstotliwości lub programowalnego woltomierza. Zamieszczane programy zostały przetestowane pod kątem typowych zastosowań. Tłumacząc je na język maszynowy można zwiększyć szybkość ich realizacji.

WOLTOMIERZ CYFROWY

Konwerter analogowo-cyfrowy (ang. **ADC**) zrealizowano z wykorzystaniem przetwornika cyfrowo-analogowego (ang. **DAC**) oraz analogowego komparatora poziomów. Program — wysyłając kolejne liczby do DAC — wytwarza na jego wyjściu coraz wyższe poziomy napięciowe, które są porównywane z mierzonym napięciem przez komparator. Zgodność napięć powoduje zmianę stanu logicznego na wyjściu komputera — co jest wykrywane przez program i powoduje jego zatrzymanie.

GENERATOR FUNKCJI

Układ pozwala generować z kwarcową stabilnością falę prostokątną, ciągi pojedynczych impulsów oraz — z pewnymi ograniczeniami — napięcie o dowolnej zależności czasowej.

Żądane przebiegi uzyskuje się za pomocą programu przesyłającego do DAC tablicę liczb odpowiadających napięciom, jakie chcemy uzyskać w kolejnych chwilach. Przybliżamy w ten sposób zadany przebieg funkcją schodkową o zbliżonym kształcie. Zamykając program w pętlę, uzyskujemy przebieg okresowy.

Generację fali prostokątnej można zrealizować znacznie prościej. Uzyskujemy ją stosując pętlę programową napisaną w języku BASIC lub w języku maszynowym. Program sprowadza się do wysyłania do portu we-wy na przemian zer i jedynek. Pętla napisana w języku BASIC daje maksymalną częstotliwość 190 Hz, podczas gdy pętla zapisana w języku maszynowym zapewnia obsługę częstotliwości do 12,5 kHz. Zmianę częstotliwości otrzymujemy wprowadzając w pętlę odpowiednie opóźnienie.

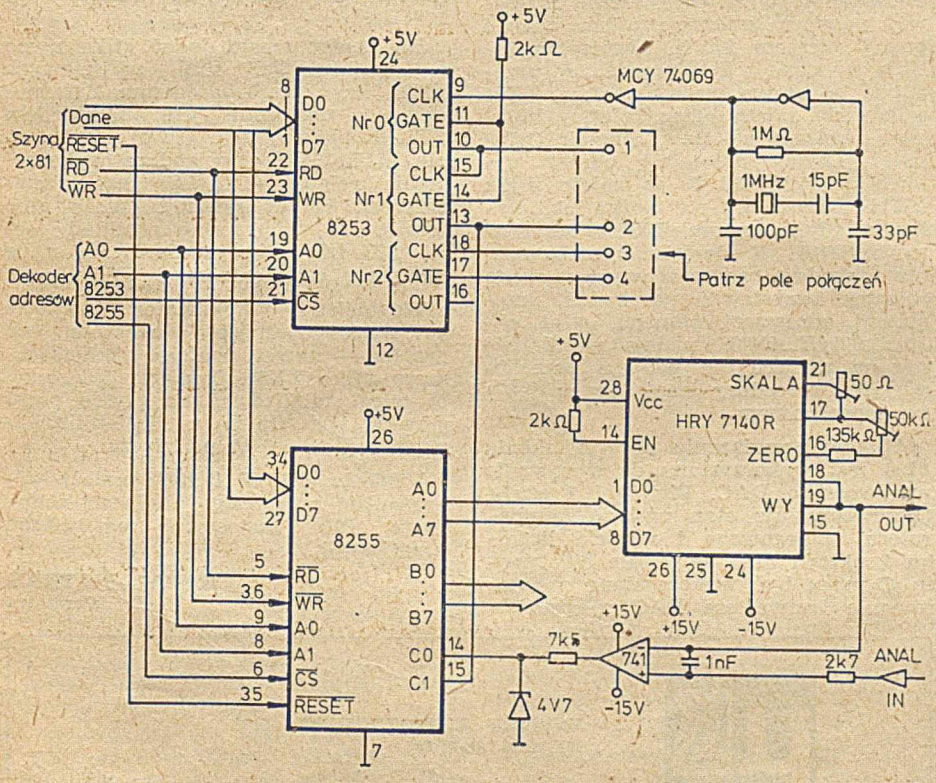
```
  4 REM POMIAR NAPIECIA
 10 POKE 8195,137
 20 FOR D=0 TO 255
 30 POKE 8192,D
 40 LET A=PEEK 8194
 50 IF A=2*INT (A/2) THEN GOTO 70
 60 NEXT D
 70 PRINT D;"V"
 80 GOTO 20
```

```
  4 REM GENERATOR PRZEBIEGU
  6 REM PROSTOKATNEGO
 10 POKE 8195,137
 20 INPUT F
 30 CLS
 40 LET N=1000000/F
 50 LET M=INT (N/256)
 60 LET L=INT ((N/256-M)*256)
 70 POKE 8199,54
 80 POKE 8196,L
 90 POKE 8196,M
100 PRINT 1000000/INT N;" HERTZ"
110 GOTO 20
```

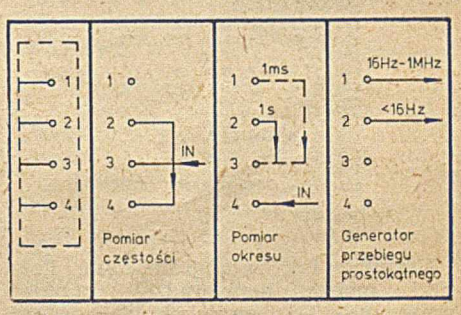
```
  1 REM 1234567890123456
  4 REM GENERACJA FALI SINUSOI-
DALNEJ
 20 FOR N=1 TO 16
 30 LET P=16513+N
 40 LET S=INT (128*(1+COS (2*PI
*N/16)))
 50 POKE P,S
 60 NEXT N
 70 FOR N=1 TO 16
 80 LET P=16513+N
 90 LET Q=PEEK P
100 POKE 8192,Q
110 NEXT N
120 GOTO 70
```

Asymetria opóźnień przełączania dla 0 i 1 daje efekt generowania impulsów dodatnich lub ujemnych. Zmiana opóźnienia za każdym obiegiem pętli pozwala na „przemiatanie" częstotliwości.

Falę prostokątną można jeszcze prościej otrzymać — korzystając ze sprzętowych udogodnień, jakich dostarcza układ 8253 zawierający trzy niezależne 16-bitowe liczniki programowane. Każdy z nich może pracować w jednym z sześciu trybów, wśród których znajdują się: tryb generacji impulsów oraz tryb generacji fali prostokątnej. Wystarczy przesłać słowo kontrolne i dwa bajty informacji do każdego z liczników, aby te — w zależności od zaprogramowanego trybu pracy — przejęły zadanie generacji impulsów czy też przebiegu prostokątnego umożliwiając ZX81 zajęcie się innymi problemami. Przebiegi wyjściowe układu 8253 powstają w wyniku podziału częstotliwości wejściowej (mogącej sięgać 2 MHz) przez



Rys. 1. Schemat ideowy przystawki



Rys. 2. Pole połączeń

zaprogramowaną wartość. Stopień podziału musi być większy od 1 a fala prostokątna przy nieparzystym współczynniku podziału ma poziom wysoki, o jeden okres zegara dłuższy od poziomu niskiego.

CZĘSTOTLIWOŚCIOMIERZ

Ta funkcja przystawki jest zrealizowana całkowicie w oparciu o własności układu 8253. Mierzona częstotliwość podawana jest na wejście licznika 2, bramkowanego impulsami o długości 1 s powstałymi z podzielenia zegara wzorcowego 1 MHz w licznikach 0 i 1 połączonych szeregowo. W tej konfiguracji liczniki 0 i 1 pracują w trybie 3, natomiast licznik 2 — w trybie 0.

Tabela 1. Format słowa kontrolnego dla 8253

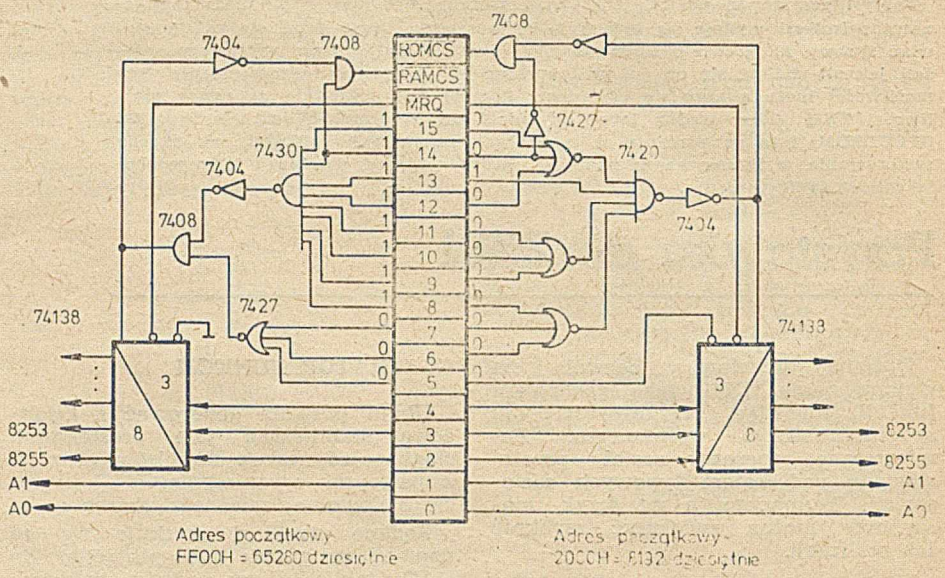
Bit	Znaczenie	Nr 0	Nr 1	Nr 5
7	nr licznika	0	0	1
6		0	1	0
5	tryb dostępu	1	1	1
4		1	1	1
3	tryb pracy	1	1	0
2		1	1	0
1		1	1	0
0	dwójkowo BCD	0	0	0
Do wpisania		54	118	176

abela 2. Format słowa kontrolnego dla 8255

Bit	Znaczenie	Dane
7	tryb/zapis C	1
6	tryb pracy	0
5		0
4	1-linie A wejście	0
3	1-linie C4÷7 wejście	1
2	tryb pracy B	0
1	1-linie B wejście	0
0	1-linie C0÷3 we'ście	1
Do wpisania		137

```
  4 REM POMIAR CZESTOSCI
 10 POKE 8195,137
 20 POKE 8199,54
 30 POKE 8196,232
 40 POKE 8196,3
 50 POKE 8199,118
 60 POKE 8197,208
 70 POKE 8197,7
 80 POKE 8199,176
 90 POKE 8198,225
100 POKE 8198,255
110 LET A=(PEEK 8194)/2
120 IF A=2*INT (A/2) THEN GOTO
110
130 LET A=(PEEK 8194)/2
140 IF NOT (A=2*INT (A/2)) THEN
GOTO 130
150 POKE 8199,128
160 LET X=PEEK 8198
170 LET Y=PEEK 8198
180 CLS
190 PRINT 65536-Y*256+X;" HERTZ
"
200 GOTO 80
```

Stosując tak zaprogramowane liczniki można doprowadzić impulsy wzorcowe do wejścia zliczającego licznika 2. Uzyskujemy w ten sposób możliwość pomiaru czasu trwania impulsu doprowadzonego na wejście bramkujące licznika 2. Ponieważ w układzie 8253 liczniki liczą „wstecz", należy pamiętać o konieczności wpisania (przed pomiarem okresu) liczby większej od spodziewanego czasu trwania impulsu, do licznika 2. Różnica między stanem końcowym a początkowym tego licznika da długość mierzonego impulsu.



Rys. 3. Dekoder adresów

KONSTRUKCJA PRZYSTAWKI

Wszystkie rejestry przystawki umieszczone są w przestrzeni adresowej pamięci, zajmując osiem pozycji ponad adresem 2000H. Miejsce to zostało wybrane z tego powodu, że przestrzeń adresowa we-wy jest w ZX81 całkowicie zajęta przez rejestry układu zarządzającego (tzw. Sinclair Computer Logic), natomiast adresy pamięci powyżej 32 K są zajęte przez MEMOPAK-64 K i nie można ich „wyłączyć". Posiadaczom ZX'ów bez rozszerzenia 64 K doradzam zastosowanie dekodera adresów wybierającego rejestry przystawki na końcu przestrzeni adresowej pamięci, gdyż obszar 2000H...4000H nadaje się świetnie na miejsce dla pamięci przechowującej procedury napisane w języku maszynowym.

W tym rozwiązaniu port A układu 8255 jest wykorzystany do sterowania przetwornika DAC. Bity 0 i 1 portu C odwzorowują stan wyjścia komparatora analogowego i wyjścia licznika 1 układu 8253. Pozostałe bity portu C mogą być wykorzystane do sterowania konwertera DAC o rozdzielczości większej niż 8 bitów. Port B jest całkowicie niewykorzystany i może być użyty do innych celów.

Pomiary częstotliwości i okresów dokonywane za pomocą tej przystawki będą tym dokładniejsze, im bardziej precyzyjny będzie zegar wzorcowy sterujący układem 8253. ZX81 jest wyposażony w generator kwarcowy, jednak jego częstotliwość (3,75 MHz) nie jest zbyt wygodna do celów pomiarowych. Z tego powodu przystawka została wyposażona w prosty generator zrealizowany na inwerterze CMOS z kwarcowym rezonatorem 1 MHz.

Na koniec warto dodać kilka uwag o obciążalności linii wyjściowych ZX81. Szyny mikroprocesora Z80 mogą być obciążane jednym układem TTL i pojemnością 130 pF. Dołączenie MEMOPAK'a powoduje wykorzystanie połowy (TTL-LS) jego obciążalności. W tej sytuacji należy albo zastosować w przystawce układy serii TTL-LS, albo zbuforować szynę adresową wzmacniaczami TTL.

Wewnętrzny stabilizator napięcia 5 V mikrokomputera znacznie się grzeje — nawet podczas obsługi samego tylko ZX81. Napięcie 5 V warto więc stabilizować na płytce przystawki, korzystając z niestabilizowanego napięcia 9 V na złączu.

Napięcia +15 V i 15 V należy doprowadzić z osobnego zasilacza sieciowego lub zastosować odpowiedni przetwornik.

RAFAŁ PIETRAK
Warszawa

μK
prowadzi
ANDRZEJ J. PIOTROWSKI
tel. dom. 48-22-85

Mikrokomputer MERITUM-I

Produkcję MERITUM-I podjęły Zakłady Urządzeń Komputerowych MERA-ELZAB w Zabrzu. MERITUM-I zostało opracowane [4] po przeprowadzeniu analizy wielu popularnych w krajach zachodnich komputerów osobistych oraz po uwzględnieniu dostępności w naszych warunkach niezbędnych podzespołów elektronicznych. MERITUM-I zbliżone jest od strony sprzętowej do TRS-80/II firmy TANDY — RADIO-SHACK, natomiast oprogramowanie jest podobne do jednej z wersji języka BASIC opracowanej przez firmę MICROSOFT.

MERITUM-I nie jest systemem modularnym, lecz wytwórca przewiduje możliwość jego rozbudowy. Adresowany jest do szerokiego kręgu odbiorców jako prosty mikrokomputer o wszechstronnym zastosowaniu.

MERITUM-I został zbudowany z podzespołów elektronicznych produkowanych w krajach RWPG. Zastosowano w nim mikroprocesor U880D (NRD-owski odpowiednik Z-80). Pamięć operacyjna RAM składa się z 16 KB zbudowanych z radzieckich pamięci dynamicznych 16K x 1 typu K 565 RU3 oraz z 1 KB pamięci statycznej (odpowiednik 2114). Oprogramowanie podstawowe może zajmować do 14 KB ROM, z czego 12 KB wykorzystuje rezydujący interpreter języka BASIC. Oprogramowanie fabryczne zawarto w pamięciach EPROM typu K573 RF2. Pamięć obrazu (video-RAM) zajmuje 1 KB.

Funkcje wejścia-wyjścia realizowane są z wykorzystaniem krajowych odpowiedników układów z rodziny 8080. Sprzęg szeregowy odpowiada standardowi RS 232C i możliwościom układu 8251. Zaleca się wykorzystanie go do sterowania drukarką (np. typu D-100). Sprzęg równoległy zrealizowano na MCY 7855 (odpowiednik 8255) — 22 bity są dostępne dla użytkownika. Sprzętowa realizacja zależności czasowych jest możliwa dzięki zastosowaniu układu zliczającego odpowiadającego 8253.

Zewnętrzną pamięcią masową jest standardowy magnetofon kasetowy. Częstotliwość sygnału zapisu wynosi 1 kHz, lecz efektywna prędkość transmisji wynosi tylko 500 bodów. W praktyce zapis na taśmę całej pojemności pamięci RAM trwa prawie 5 minut. Przyjęty sposób zapisu nie zawiera nadmiarowości kodu, a więc samoczynna korekta błędu nie jest możliwa. Ryzyko posługiwania się błędnym zapisem lub odczytem zmniejsza zastosowanie sumy kontrolnej oraz możliwość programowego sprawdzenia zapisu.

Mikrokomputer MERITUM-I generuje zespolony sygnał wizyjny, co umożliwia przyłączenie monitora telewizyjnego (np. typu NEPTUN 156 lub MTU 3159). Przyłączenie telewizora jest możliwe jedynie za pośrednictwem dodatkowego adaptera (modulator pracujący w kanale 1-6). Ekran jest zorganizowany w 16 wierszy, a liczbę znaków w wierszu można programowo ustawić na 32 lub 64. Oprócz znaków alfanumerycznych można wyświetlać 64 specjalne symbole graficzne. Ekran może być również traktowany jak zbiór dostępnych programowo punktów (128 poziomo i 48 pionowo).

Klawiatura kontaktronowa w układzie QWERTY obsługiwana jest programowo. Uzupełniają ją klawisze specjalne:

RESET: do zerowania całego mikrokomputera, w tym pamięci i portów

NMI: do zerowania mikrokomputera bez zerowania pamięci. Wykorzystywany w przypadku „zawieszenia się" systemu, gdy zależy nam na zachowaniu programu w pamięci RAM

CLEAR: zeruje ekran monitora

ENTER: kończący wpisywanie linii programu, zleceń systemowych itp.

BREAK: przerywający wykonywanie programu.

Układy mikrokomputera umieszczone są w obudowie o wymiarach: 380 × 250 × 90 mm, razem z klawiaturą. Zasilacz sieciowy znajduje się w podobnej (lecz węższej — 120 mm) obudowie i dostarcza napięć: +5 V/3 A, +12 V/1 A oraz —5V/0,2 A. Obudowy wykonano z tworzywa sztucznego.

MERITUM-I posiada (w pamięci EPROM) interpreter języka BASIC. Poniżej opiszemy niektóre cechy charakterystyczne dla zastosowanej wersji tego języka.

Nazwy zmiennych i tablic są rozpoznawane przez dwa pierwsze znaki, lecz mogą być dłuższe. Istnieją cztery typy zmiennych. Mogą one być określone przez definiowanie lub przez dodanie do nazwy odpowiedniego symbolu.

DEFINT (%) — dla zmiennych całkowitych. Zajmują one w pamięci po 2 bajty i mają zakres od —32768 do +32767.

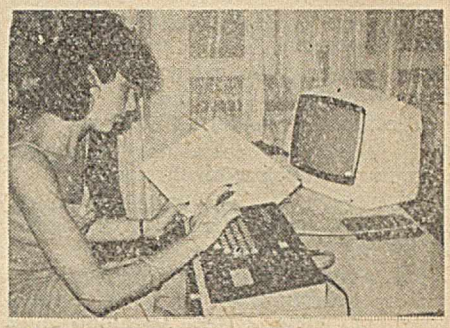
DEFSGN (!) — dla zmiennych o pojedynczej precyzji. Zajmują one w pamięci po 4 bajty i obejmują zakres liczbowy $\pm 1{,}7 \cdot 10^{38}$, dając 6 cyfr znaczących.

DEFDBL (#) — dla zmiennych podwójnej precyzji. Zajmują one w pamięci po 8 bajtów i obejmują zakres liczbowy $\pm 1{,}7 \cdot 10^{38}$, dając 16 cyfr znaczących.

DEFSTR ($) — dla zmiennych łańcuchowych. Mogą one mieć długość do 255 znaków, a każdy znak zajmuje 1 bajt pamięci.

Instrukcje CDBL i CSNG pozwalają na konwersję typu zmiennej między pojedynczą i podwójną precyzją.

Najmniejszą wartością indeksu w tablicy jest zero. Tablice mogą być wielowymiarowe. Brak jest jednak



MERITUM-I z monitorem NEPTUN 156
Fot. Zbigniew Chrześcijański

standardowych działań na nich (np. funkcji macierzowych).
Dostępne są natomiast standardowe operacje na zmiennych łańcuchowych. Są również funktory logiczne AND-OR-NOT.

Brak jest funkcji definiowanych przez użytkownika. W rezultacie nie ma możliwości przekazywania w ogólny sposób. parametrów do procedur Zmusza to do stosowania różnych „chwytów".

Instrukcje sterujące mają wygodną postać: **IF-THEN-ELSE** oraz **ON X GOTO N_1, N_2,...,N_k** i **ON X GOSUB N_1, N_2,...,N_k**.
Nie ma operacji na plikach, co jest konsekwencją wykorzystywania magnetofonu jako pamięci zewnętrznej. Można jednak rozdzielić w kasecie programy i dane tak przy zapisie, jak i przy odczycie.

Użyteczne są instrukcje obsługi błędów:

ON ERROR GOTO — skok do podprogramu z obsługą błędu

RESUME — koniec podprogramu obsługi błędu

ERL — wyznaczenie numeru linii, w której wystąpił błąd

ERR — wyznaczenie kodu zaistniałego błędu

ERROR — symulacja błędu do badania reakcji programu.

Dzięki nim błędy nie muszą przerywać wykonywania programu, a jeśli nie można tego uniknąć, to wznowienie pracy może być szybsze.

Gospodarowanie pamięcią ułatwiają instrukcje:

MEM — wyznacza liczbę wolnych bajtów w pamięci

FRE — wyznacza wielkość obszaru przeznaczonego na łańcuchy

VARPTR — określa adres zmiennej w pamięci.

Instrukcje **PRINT i PRINT TAB** są uzupełnione przez:

POS — badanie aktualnej pozycji kursora

PRINT @ — wpisanie znaku na wskazaną pozycję ekranu; ekran składa się z 1024 komórek utworzonych przez 16 linii po 64 znaki

PRINTUSING — drukowanie tekstu według określonego wzorca.

Dostępne są następujące instrukcje obsługi ekranu:

CLS — zerowanie całego ekranu

POINT (x,y) — testowanie punktu graficznego o współrzędnych x,y

SET (x,y) — rozjaśnienie punktu ekranu o współrzędnych x,y

RESET (x,y) — zaciemnienie punktu graficznego o współrzędnych x,y gdzie $0 \leqslant x \leqslant 127$ i $0 \leqslant y \leqslant 47$.

Do wywoływania podprogramów w języku maszynowym służy instrukcja USR (x), gdzie x jest parametrem z zakresu —32768 do +32767. Wynik działania procedury może być przekazany do programu w języku BASIC. Programy w języku maszynowym również mogą być ładowane z magnetofonu i uruchamiane do wskazanego adresu.

Komunikacja MERITUM z otoczeniem jest ułatwiona dzięki instrukcjom: INP (wczytuje bajt danych ze wskazanego portu) oraz OUT (wysyła dane do podanego portu).

Kilkanaście poleceń operatorskich umożliwia poprawianie programu. Jednak proces redagowania może odbywać się tylko w zakresie jednej linii programu. Duże znaczenie praktyczne mają polecenia TRON i TROFF, pozwalające na śledzenie przebiegu uruchamianego programu.

WSTĘPNA OCENA MIKROKOMPUTERA

O wartości mikrokomputera decyduje realnie dostępne oprogramowanie oraz jakość dokumentacji systemu. Pierwsze uwagi użytkowników MERITUM dotyczą znacznego nagrzewania się mikrokomputera i zasilacza, zwłaszcza podczas dłuższej pracy. Spodziewamy się, że może to być przyczyną różnych awarii i zakłóceń w pracy.

Dokumentacja techniczno-ruchowa mikrokomputera (31 stron formatu A-5) jest źle opracowana. Jej treść pozwala jedynie na bardzo powierzchowne zapoznanie się z urządzeniem. Rysunki i schematy włączono do instrukcji serwisowej, a producent od razu zastrzega, że nie będzie ona udostępniana posiadaczom MERITUM-I.

Ani opis sprzętu, ani opis oprogramowania nie pozwalają na zapoznanie się z programowaniem w języku wewnętrznym. Informacje o możliwościach dostępu do zasobów sprzętowych (np. układów we-wy) ograniczają się do krótkiej tabeli podającej adresy portów. Użytkownik odsyłany jest do katalogów firm zachodnich. Znaleźliśmy też szereg niejasności czy wręcz sprzeczności w szczegółach dokumentacji:

— na str. 5 podano, że interfejs magnetofonu jest asynchroniczny, a na str. 12 pokazano sygnał zapisu z bitami synchronizacji,

— na str. 21 opis wtyku do podłączenia monitora nie jest zgodny ani z gniazdem w monitorze, ani z tym co dostaje odbiorca,

(W szczególności nie wyprowadzono sygnału fonii.)

— na str. 5 wymieniono „repertuar" znaków semigraficznych — 96, repertuar znaków graficznych — 128". Podręcznik oprogramowania na str. 89 podaje tylko 64 znaki graficzne,

— nie podano opisu sygnałów na złączu interfejsu równoległego.

Dokumentacja oprogramowania to opis języka BASIC-MERITUM na 89 stronach A-5 z niewielką liczbą przykładów. Jest on wydrukowany starannie — lecz niestety, jego przejrzystość i komunikatywność nie jest zadowalająca. Opis języka jest bardzo powierzchowny i nie zawiera wielu istotnych informacji — choćby liczby wymiarów tablic, zakresu indeksów w tablicach itd. Bardzo przydałby się opis pamięci ROM — np. adresy początkowe i charakterystyka procedur oraz bliższe dane o zmiennych i tablicach systemowych itp. Nie dostarczono nam kasety z kursem programowania, mimo że jest ona przewidziana do kompletu urządzenia. Nie wiadomo dziś, czy będzie dostępne jakiekolwiek dodatkowe oprogramowanie — choćby w BASICU.

ZBIGNIEW CHRZEŚCIJAŃSKI
JACEK ŻEBROWSKI
Łódź

LITERATURA

[1] Dokumentacja techniczno-ruchowa mikrokomputera MERITUM-I nr ewid. 44990000

[2] Podręcznik programowania i użytkowania BASIC-MERITUM nr ewid. 44990001

[3] Korga Z., Smoliński A., Lipowski J., Podsiadło P.: Komputer osobisty MERITUM. Biuletyn Techniczno-Informacyjny MERA, nr 10/1983, s. 3

[4] Korga Z.: Mikrokomputer osobisty — MERITUM. Biuletyn Techniczno-Informacyjny MERA, nr 5—6/1984, s. 52.

Wracamy do początkowo zakładanej koncepcji kącika dla miłośników FORTHA. Pauza, choć nie do końca zaplanowana przez redakcję, umożliwiła przekazanie czytelnikom zasobu „szkolnych" informacji poprzez obszerne publikacje w części artykułowej INFORMATYKI. Dzięki temu możemy teraz odstąpić od opisywania podstaw FORTHA i zająć się publikowaniem ciekawych procedur (czyli tzw. słów). W poniższym tekście nawiązujemy do opisu implementacji FIG-FORTHA opisując słowo pozwalające na wprowadzenie pewnej użytecznej modyfikacji.

Zapraszamy amatorów FORTHA do nadsyłania opisów własnych słów, które mogą okazać się przydatne dla innych Czytelników.

FORTH (5)

PROSTE PROGRAMY NARZĘDZIOWE

LOOK-IN — „podglądanie" zawartości słowa zapisanego w Słowniku (dla słów definiowanych w FORTH bez ASSEMBLERA).

Słowa pomocnicze:
(wszystkie słowa definiowane przy użyciu HEX)

FIND — odszukanie CFA dla słowa podanego w łańcuchu wejściowym. Słowo to występuje w standardzie FORTH 79 lecz brak go w niektórych implementacjach. Na przykład w FIG-FORTH (CP/M) występuje —**FIND.**

```
: FIND  ( DLA FIG-FORTH )
        -FIND
        IF DROP 2 -
        ELSE ." BRAK SLOWA"
             BELL        ( : BELL 7 EMIT ; )
        ENDIF ;

: FIND-PFA      ( JAK FIND LECZ PFA )
        FIND 2 + ;

: NFA-FOR-CFA   ( ODSZUKANIE NFA PRZY ZNANYM CFA )
        3 - BEGIN 1 - C@ 80 > UNTIL ;

: NAME? ( CZY ADRES SLOWA ZDEFINIOWANEGO W FORTHIE )
        DUP NFA-FOR-CFA DUP C@
        80 - + 3 + = ;

: NAME(CFA)     ( PODANIE NAZWY SLOWA O ZNANYM CFA )
        NFA-FOR-CFA
        ID.     ( IDDOT WYSWIETLA SLOWO ) ;

: NAME(PFA)     ( PODANIE NAZWY SLOWA O ZNANYM PFA )
        2   NAME(CFA) ;

: L1    ( SLOWO CZESCIOWE DLA LOOK-IN, PODAJE ROZWIJANE SLOWO )
        FIND-PFA CR DUP ." : " NAME(PFA) CR ;

: L2    ( SLOWO CZESCIOWE DLA LOOK-IN )
        (PODAJE NAZWE KOLEJNEGO SKLADNIKA )
        DUP @ OVER ." ADR=" . ." HEX=" DUP . DUP
        ." NAME=" NAME? IF NAME(CFA) ELSE ." ????" DROP ENDIF ;

: ;S?   ( SLOWO CZESCIOWE DLA LOOK-IN )
        ( SPRAWDZA CZY KONIEC DEFINICJI )
        2 + DUP @ 2 +     ( NASTEPNY ADRES )
        ' ;S = ;          ( ' ODKLADA PFA SLOWA ;S )
                          ( KONCZACEGO DEFINICJE )
```

```
: (.")  ( CZY NA STOSIE JEST CFA SLOWA ." ? )
        DUP @ 2 +
        ' (."). ( ' DOKLADA PFA DLA NASTEPNEGO SLOWA )
        = ;

: "?    ODSZUKUJE KONIEC NAPISU DLA ." )
        DUP 2 + @ FF AND        ( OBLICZENIE DLUGOSCI )
        SWAP OVER 1 + OVER + ROT ROT 2 + ;

: E00FF ( WYSWIETLENIE ZNAKU Z LEWEGO BAJTU )
        DUP @ FF AND EMIT ;

: EFF00 (WYSWIETLENIE ZNAKU Z PRAWEGO BAJTU )
        DUP @ FF00 AND 100 / EMIT ;


: E."   ( WYSWIETLENIE NAPISU )
        SWAP 1 - SWAP EFF00 2 + ( PIERWSZY ZNAK )
        OVER IF BEGIN E00FF OVER 1 -    ( LEWY BAJT )
                IF EFF00 ENDIF  ( PRAWY BAJT )
                2 + SWAP 2 - SWAP OVER
                1 < UNTIL
             ENDIF DROP DROP ;


: LOOK-IN       ( PODGLADANIE ZAWARTOSCI ZDEFINIOWANEGO SLOWA )
        BASE @  ( ZAPAMIETANIE AKTUALNEGO SYSTEMU )
        HEX     ( USTALENIE SYSTEMU SZESNASTKOWEGO )
        L1      ( NAZWA ROZBIERANEGO SLOWA )
        BEGIN
                L2      ( NAZWA KOLEJNEGO SKLADNIKA )
                (.")?   ( CZY NAPIS ? )
                IF "? E." ENDIF CR      ( WYSWIETLENIE NAPISU )
                ;S?     ( CZY KONIEC DEFINICJI )
        UNTIL
        ." ; " CR DROP  ( KONIEC SLOWA )
        BASE !  ( POWROT DO STAREGO SYSTEMU )
```

PRZYKŁAD ZASTOSOWANIA LOOK-IN

System operacyjny CP/M staje się w kraju standardem dla mikrokomputerów 8-bitowych. Coraz popularniejszy staje się jednocześnie FIG-FORTH, implementacja FORTHA pod kontrolą systemu CP/M.

Przykład zastosowania procedury LOOK-IN jest jednocześnie receptą na drobną modyfikację FIG-FORTHA. Wersja standardowa zakłada, że konsola komunikuje się z systemem słowami o długości 7 bitów. Procedura **ID. (iddot)** przesyłająca nazwę słowa według zadanego NFA (adresu pola nazwy), nie obcina najbardziej znaczącego bitu w ostatnim znaku nazwy zdając się na odpowiednią obsługę sprzętową. Z tego powodu w nowszych konstrukcjach mikrokomputerów, gdzie układ wyświetlający dysponuje możliwościami graficznymi i wykorzystuje pełny bajt przy komunikacji z CPU — procedura **ID.**, a także **VLIST** (oparta na ID.) nie działa poprawnie.

Aby usunąć tą niedogodność wystarczy w **iddot** zastąpić **TYPE** przez własną procedurę TYPE-ASC.

```
: TYPE-ASC      ( zerowanie najbardziej znaczącego bitu i TYPE )
        OVER OVER  ( powielenie parametrów TYPE )
        0 DO
           DUP@ 7F7F AND  ( pobranie, odrzucenie bitu )
           OVER ! 1+      ( i zapisanie ponownie )
        LOOP DROP
        TYPE  ( wyświetlenie )
```

Przykład - modyfikacja FIG-FORTH (podane adresy są przykładowe)

1. Znalezienie adresu (CFA) dla TYPE-ASC:

```
FIND TYPE-ASC . ( cr ) 267F OK
```

2. Odszukanie TYPE w ID. :

```
1 3 ! ( przełączenie systemu na drukarkę wg CP/M )
LOOK-IN ID. ( cr )
```

(spodziewany wydruk):

```
: ID.
adr=E4A   hex=CD2    name=PAD
adr=E4C   hex=156    name=LIT
adr=E4E   hex=20     name=????
adr=E50   hex=156    name=LIT
adr=E52   hex=5F     name=????
adr=E54   hex=C7A    name=FILL
adr=E56   hex=556    name=DUP
adr=E58   hex=950    name=PFA
adr=E5A   hex=91C    name=LFA
adr=E5C   hex=52D    name=OVER
adr=E5E   hex=847    name=-
adr=E60   hex=CD2    name=PAD
adr=E62   hex=549    name=SWAP
adr=E64   hex=31D    name=CMOVE
adr=E66   hex=CD2    name=PAD
adr=E68   hex=AE7    name=COUNT
adr=E5A   hex=156    name=LIT
adr=E6C   hex=1F     name=????
adr=E6E   hex=3C3    name=AND
adr=E70   hex=AFA    name=TYPE
adr=E72   hex=8C1    name=SPACE
;
OK
```

Wracamy do klawiatury:

```
0 3 ! (cr) OK
```

i wpisujemy CFA dla TYPE-ASC w miejsce CFA TYPE:

```
267F E70 ! (cr) OK
```

Teraz już tylko dla sprawdzenia:

```
VLIST (cr)
```

MAREK CZARZYŃSKI
Warszawa

Żyrardowskie Zakłady Tkanin Technicznych

96-300 ŻYRARDÓW, ul. Okrzei 51

oferują do sprzedaży:

- pamięć półprzewodnikową do minikomputera MERA 400 o pojemności 32 K słów
- urządzenia do obsługi taśmy papierowej:
  - czytnik CT 2100 M1
  - perforator DT 105 S
  - rozwijacz RT 2200
  - zwijacz ZT 2200
- dziurkarki kart Soemtron 415–1
- sprawdzarkę kart Soemtron 425–1

Informacji udziela Dział Informatyki, tel. 20-31 do 35 w. 284

EO/142/K/85

Wbrew wszelkim poczynaniom, które „nie pomagają" w rozwoju mikroinformatyki, coraz więcej osób zaczyna się w Polsce posługiwać mikrokomputerami. Najbardziej popularnego dla mikrokomputerów — języka BASIC — można się nauczyć z instrukcji obsługi. Mówiąc precyzyjnej, nowicjusz może dowiedzieć się z instrukcji obsługi jakie jest działanie poszczególnych instrukcji i — zacząć programować. Aby jednak programować dobrze, trzeba mieć za sobą lata doświadczeń lub... skorzystać z doświadczeń innych.

W Akademii mikroKLANU nie piszemy jak programować, lecz — jak lepiej programować. Wychodzimy przy tym z założenia, że dobrze napisany program zawsze można kosztem odstępstwa od publikowanych reguł dopasować jakoś do mniejszych zasobów pamięci w komputerze. Natomiast źle napisany program, choćby zajmował minimalną ilość miejsca w pamięci, szybko idzie do lamusa.

Bardziej doświadczonych programistów zapraszamy do przedstawienia własnych refleksji.

Akademia MikroKLANU (2)

Silnia

W pierwszym odcinku omawiane było wyjście z programu. Przede wszystkim apelowaliśmy o pełne wydruki. Mamy jednak nadzieję, że Czytelnicy nie potraktowali naszych rad dosłownie — spotyka się wydruki składające się z kilkudziesięciu stron, gęsto zapisane liczbami. Takie wyjście jest gorsze niż w przypadku zbyt skąpej informacji: użytkownik programu sam musi spełniać rolę komputera i wyszukiwać potrzebne mu liczby — niczym igłę w stogu. Nieco przewrotnie przedstawiamy więc program ilustrujący poprawne wejście, który drukuje bardzo dużo cyfr — ale takie jest jego przeznaczenie!

Zasady organizacji wejścia do programu są proste: należy przedrukować wszystko, co zostało wprowadzone do komputera. Jest to najlepszy sposób kontroli poprawności wprowadzania danych, szczególnie jeżeli program — tak dzieje się w naszym przykładzie — długo działa. Zasadą o mniejszym znaczeniu jest reguła automatycznego wprowadzania wszystkich stałych oraz typowych wyborów. Dotyczy to w szczególności wyboru drukarki lub monitora ekranowego jako urządzenia wyjścia. Program, który korzysta z obu możliwości, powinien zapytać użytkownika o rodzaj wyjścia, sugerując bardziej prawdopodobne lub wygodniejsze urządzenie. W naszym przykładzie nie mamy takich problemów. Warto natomiast zwrócić uwagę, że program sugeruje jaka może być największa liczba, dla której obliczenia mogą być jeszcze wykonane przy danym wymiarze wektora A. Taka konstrukcja programu (mimo uwag wstępnych, patrz linie 5 oraz 6) pozwala go wykorzystywać również tym, którzy nie znają treści programu (użytkownik wcale nie musi umieć programować). Zasada, którą tu zastosowaliśmy, opiera się na zwielokrotnieniu informacji ważnych z punktu widzenia operacyjnego.

Ważną sprawą, która pojawia się podczas analizy przykładowego programu, jest także optymalizacja obliczeń. Szczegółowe uwagi o takiej optymalizacji — zarówno z punktu widzenia wykorzystania pamięci komputera, jak i szybkości obliczeń — można znaleźć w świetnej książce D. van Tassela „Praktyka programowania", WNT, 1978. W naszym programie wewnętrzna pętla (po J) będzie powtarzana wielokrotnie razy (dla dużych N). Każde uproszczenie programu w tej pętli pozwoli skrócić (długi!) czas jego realizacji. Dlatego też, mimo mniejszej czytelności tak zaprogramowanego wyrażenia, zdecydowaliśmy się zastąpić dzielenie przez dziesięć mnożeniem przez 0.1. Zamiana ta rzeczywiście redukuje czas obliczeń, gdyż dla każdego komputera dzielenie jest operacją dłuższą niż mnożenie. Podobne reguły dla innych działań można znaleźć we wspomnianej książce Tassela.

A oto nasz program szkolny II — służy on do obliczania silni dużych liczb z pełną dokładnością, tzn. podaje wszystkie cyfry. Zwróćmy uwagę, że standardowy BASIC nie pozwala wyprowadzać informacji w instrukcji INPUT — dlatego też najpierw drukujemy pytanie o liczbę, której silnię będziemy obliczali, a dopiero potem występuje instrukcja wejścia. Dla większości znanych mikrokomputerów instrukcje te można połączyć w jedną:

```
20 INPUT "LICZBA (<1001)";N
```

Jednak pisząc program dla wszystkich musimy trzymać się reguł składni języka BASIC. Trochę to niewygodne, ale dzięki temu nasz program może być uruchomiony na każdym komputerze wyposażonym w interpreter lub kompilator BASICA!

```
5 REM OBLICZANIE SILNI BARDZO DUZYCH LICZB
6 REM (N<1001) Z PELNA DOKLADNOSCIA.
10 DIM A(3000)
20 PRINT "LICZBA (<1001)?"
21 INPUT N
30 PRINT N;"!=";
40 D=1
50 C=0
60 A(1)=1
70 FOR I=2 TO N
80 FOR J=1 TO D
90 A(J)=A(J)*I+C
100 C=INT(0.1*A(J))
110 A(J)=A(J)-10*C
120 NEXT J
130 IF C=0 THEN GOTO 190
140 C1=INT(0.1*C)
150 D=D+1
160 A(D)=C-10*C1
170 C=C1
180 GOTO 130
190 NEXT I
200 FOR I=D TO 1 STEP -1
210 PRINT A(I);
220 NEXT I
230 END
READY.
    RUN
LICZBA (<1001)?
? 15
 15 != 1 3 0 7 6 7 4 3 6 8 0 0 0
READY.
```

Zwróćmy jeszcze uwagę, że dla zupełnie głupiego użytkownika (tacy też się, niestety, zdarzają) można by dodać linię 25:

```
25 IF N <1 OR N> 1001 THEN GOTO 20
```

która radykalnie zabezpieczy przed niewłaściwym wejściem. Ponieważ nasz program ma bardzo długi czas realizacji, więc wykrycie, że wprowadzona liczba N jest większa niż 1001 w pierwotnej wersji programu nastąpi dopiero pod koniec obliczeń! Czy zastosujemy tak radykalne zabezpieczenie w programie, który piszemy tylko dla siebie? Decyzja powinna zależeć przede wszystkim od grona potencjalnych użytkowników — tak naprawdę nigdy nie piszemy żadnego programu tylko dla siebie. Łatwo przekonać się o tym, gdy będziemy dysponowali dobrymi programami — na pewno wielu znajomych będzie chciało się z nami wymieniać.

Życząc Czytelnikom szerokiego rozpowszechnienia ich wspaniałych programów kończymy posiedzenie Akademii.

JAKUB TATARKIEWICZ

Ośrodek Obliczeniowy PPiMUEB „Elektromontaż"

70-370 Szczecin, ul. Bohaterów Warszawy 15/16
tel. 450-02 w. 26 lub 78, teleks 0422210

zakupi:

dwa minikomputery MERA 100

w dowolnej konfiguracji

EO/138/K/85

dokończenie ze str. 12

Formaty rozszerzone zależą od implementacji i stanowią podstawowe narzędzie programowania operacji pośrednich (np. w obliczaniu funkcji elementarnych). Umożliwiają one osiągnięcie zakładanej przez normę dokładności obliczeń. Norma stanowi bowiem, że wynik obliczenia może mieć co najwyżej błąd jednego zaokrąglenia. Znaczy to, że w praktyce obliczenia muszą być prowadzone przy użyciu formatów szerszych niż podstawowe, a następnie — zaokrąglane.

Liczba w **formacie rozszerzonym pojedynczej precyzji** składa się z czterech elementów:
— jednobitowego znaku s
— wykładnika e
— jednobitowej części całkowitej liczby j (umożliwia to denormalizację)
— mantysy f (o długości co najmniej 32 bity).
Minimalny zakres zmian wykładnika wynosi od —1023 do 1024. Wartość liczby:
— jeśli e=M (największa wartość wykładnika) i f≠0, to v=NaN
— jeśli e=M i f=0, to $v=(-1)^s\cdot\infty$
— jeśli m<e<M (m — najmniejsza wartość wykładnika), to $v=(-1)^s 2^e(j.f)$
— jeśli e=m i j=f=0, to $v=(-1)^s 0$ (czyli zero)
— jeśli e=m, a j lub f nie jest zerem, to $v=(-1)^s 2^e(j.f)$, przy czym e'=m lub m+1, zależnie od implementacji.

Format rozszerzony podwójnej precyzji, podobnie jak dla pojedynczej, składa się z czterech elementów. Wykładnik musi mieć zakres co najmniej od —16 383 do 16 384, a część ułamkowa musi mieć co najmniej 63 bity.

Norma przewiduje, że każda implementacja musi zawierać przynajmniej jeden format pojedynczej precyzji oraz powinna zawierać co najmniej jeden format rozszerzony, odpowiadający szerokością najdłuższemu formatowi głównemu.

ZAOKRĄGLENIA

W normie przewidziano różne możliwości kontrolowania przebiegu obliczeń i różne sposoby zaokrąglania wyników. Najbardziej istotne i chyba najważniejsze w całej normie jest stwierdzenie, że poza konwersją między postacią dwójkową a postacią BCD [1] — wszystkie operacje arytmetyczne (włącznie z operacjami na nie-liczbach NaN) muszą być wykonywane tak, jakby ich wyniki miały być dokładne, a następnie zaokrąglone do końcowego formatu.

Wynika stąd, że jedynym dopuszczalnym błędem wyniku może być błąd zaokrąglenia. Tak mocne stwierdzenie nakłada dość ostre warunki na implementację, tym bardziej, że norma przewiduje różne sposoby zaokrąglania wyników — opisane poniżej.

● **Zaokrąglanie do liczby najbliższej** (domyślne). Wynik operacji może różnić się od wartości dokładnej co najwyżej o 1/2 wartości ostatniego bitu. Gdy różnica ta wynosi dokładnie 1/2, ostatni bit musi być jedynką.

● **Zaokrąglanie ukierunkowane.** Implementacja normy musi umożliwiać wybór zaokrąglania w kierunku: liczb większych, liczb mniejszych i zera — dla każdej operacji przewidzianej normą. W pierwszym przypadku wynik zaokrąglania jest liczbą najbliższą wynikowi dokładnemu, ale nie mniejszą niż wynik dokładny (z wyjątkiem sytuacji gdy wynikiem jest ∞). W drugim przypadku wynik zaokrąglony jest liczbą najbliższą, lecz nie większą od wyniku dokładnego. W trzecim przypadku wynik jest liczbą najbliższą i nie większą co do wartości bezwzględnej od wyniku dokładnego. Sposób zaokrąglania może mieć wpływ na znak sum równych zero.

Wynik jest zwykle zaokrąglony z dokładnością osiąganą w formacie wynikowym. Jednakże sprzęt maszyny cyfrowej może określać wynik operacji w formacie rozszerzonym i w takim formacie jest on przechowywany w pamięci. Użytkownik, np. kompilator języka wysokiego poziomu, musi mieć wtedy możliwość wskazania sposobu zaokrąglania wyniku (np. z dokładnością formatu zwykłego pojedynczej precyzji).

Główna zasada polega na tym, że wynik nie może zawierać więcej niż jednego błędu zaokrąglenia.

[1]) BCD — postać dziesiętna kodowana dwójkowo (ang. binary coded decimal)

OPERACJE STANDARDOWE

Implementacja normy musi zawierać następujące operacje: dodawanie, odejmowanie, mnożenie, dzielenie, pierwiastek kwadratowy (uwaga: $\sqrt{-0}=-0$), reszta z dzielenia przez liczbę całkowitą, konwersje liczb z dowolnych formatów na dowolne formaty (również — liczb całkowitych na zmiennoprzecinkowe i odwrotnie oraz konwersje dwójkowo-dziesiętne), porównywanie.

Operacje arytmetyczne muszą być wykonywane na wszystkich dostępnych w danej implementacji formatach. Nie jest konieczne wprowadzanie tej samej operacji dla kilku formatów, ale wtedy format wyniku operacji musi być najdłuższym zaimplementowanym formatem. Wyniki operacji winny być zaokrąglane według wyżej podanych reguł.

Operacje konwersji liczb muszą zapewniać możliwość przeprowadzenia konwersji liczby zmiennoprzecinkowej z dowolnego formatu na dowolny inny format. Jeśli format wynikowy jest krótszy, to wynik winien być zaokrąglony według wyżej opisanych reguł, w przeciwnym razie — musi być dokładny. Ponadto, powinna być przewidziana możliwość konwersji liczby na liczbę całkowitą. Obowiązują wtedy reguły zaokrąglania podane wyżej. Konwersja dwójkowo-dziesiętna musi być możliwa dla wszystkich formatów. Wyniki winny być zaokrąglane. Norma określa szczegółowo zakres wartości liczb dziesiętnych podlegających konwersji. Wymaga się przy tym, aby konwersja była monotoniczna. Nadmiar, niedomiar oraz wystąpienie nie-liczby (NaN) winny być właściwie sygnalizowane.

Operacje porównywania muszą być możliwe do wykonania we wszystkich implementowanych formatach, włącznie z porównywaniem liczb w różnych formatach. Przewiduje się istnienie czterech rozłącznych relacji: mniejszość, większość, równość i nieokreśloność. Ta ostatnia winna zachodzić, gdy co najmniej jedna z liczb jest typu NaN, lub gdy ∞ jest porównywana z jakąkolwiek liczbą różną od ∞ przy zaokrąglaniu w kierunku zera (tryb rzutowy — ang. projective mode). Znaki ∞ (+∞ lub —∞) winny być ignorowane.

NIESKOŃCZONOŚĆ, NIE-LICZBY I ZNAK ZERA

Wartości nieskończenie duże (jeśli takie występują) mogą — według normy — występować w dwóch postaciach, w zależności od potrzeby. W pierwszej postaci (domyślnej), tzw. rzutowej (ang. projective) obydwie wartości (—∞ i +∞) uważa się za równe, a wskutek porównania z dowolną (skończoną) liczbą uzyskuje się wynik nieokreślony. Druga postać, tzw. afiniczna (ang. affine), spełnia warunek:

—∞<(każda skończona liczba)<+∞

Poza operacjami niedozwolonymi dla ∞, wszystkie inne dają wynik dokładny i nie powinny powodować wyjątków. Wyjątek może powstać, gdy:

● wartość ∞ powstała wskutek operacji wykonanej na liczbach skończonych, powodującej przepełnienie lub — wskutek dzielenia przez zero, przy czym odpowiadające tej sytuacji przerwanie programowe (ang. trap) jest zabronione.

● ∞ jest argumentem nielegalnym.

Implementacja winna zawierać dwa rodzaje nie-liczb:
— wywołujące przerwanie programowe (ang. trap)
— nie wywołujące przerwania.
Rodzaj pierwszy winien spowodować ustawienie odpowiedniego wskaźnika nielegalnej operacji lub powstanie wyjątku — w zależności od implementacji. Rodzaj drugi winien umożliwiać uzyskanie informacji o niewłaściwych lub też nieosiągalnych danych lub wynikach. Każda operacja z nie-liczbą wywołująca przerwanie winna — jeśli się ono nie pojawi — ustawiać wskaźnik nielegalnej operacji i dać w wyniku nie-liczbę nie wywołującą przerwania. Jeśli argumentami operacji są dwie nie-liczby nie wywołujące przerwania, to nie jest ustawiany żaden wskaźnik, a wynikiem jest również nie-liczba.

Norma nie określa znaku dla nie-liczby. W pozostałych przypadkach zaś:
— znak iloczynu i ilorazu jest różnicą symetryczną znaków argumentów
— znak sumy lub różnicy (traktowanej jako suma, w której jeden ze składników ma znak minus) różni się od najwyżej jednego znaku składnika.
Reguły te dotyczą również sytuacji, gdy argumentami operacji są zera lub nieskończoności. Jeśli suma (lub róż-

nica) dwóch składników o znakach przeciwnych jest równa zeru, to ma ona znak plus we wszystkich sposobach zaokrąglania z wyjątkiem zaokrąglania w dół — wtedy znakiem jest minus. Wyrażenie $x+x = x-(-x)$ zachowuje znak x, nawet gdy $x=0$. Pierwiastek kwadratowy może mieć wartość ujemną, gdy jego argumentem jest —0.

ARYTMETYKA NIEUNORMOWANA I NIEZNORMALIZOWANA

Jeżeli choć jeden z argumentów operacji zmiennoprzecinkowej nie jest znormalizowany, norma przewiduje domyślne (ang. default) tryby wykonywania takich operacji:

● **Dodawanie i odejmowanie.** Jeśli jeden ze składników jest znormalizowany, to wynik musi być znormalizowany przed zaokrągleniem. Jeśli nie — to wykładnik sumy jest równy największemu wykładnikowi składników.

● **Mnożenie.** Wykładnik jest równy sumie wykładników.

● **Dzielenie.** Wykładnik jest równy różnicy wykładników zmniejszonej o 1, gdy dzielnik jest różny od zera i znormalizowany. Jeśli tylko jeden z argumentów jest nieznormalizowany, drugi zaś równy zeru lub ∞, to wynik jest taki sam jak w przypadku, gdy nieznormalizowany argument byłby znormalizowany.

● **Reszta z dzielenia całkowitoliczbowego (z:=xREMy).** Wynik (z) musi być obliczony tak, jakby argument x był najpierw znormalizowany.

● **Pierwiastek kwadratowy.** Jeśli argument jest nieznormalizowany, to operacja jest nielegalna.

● **Konwersja (z:=x).** Wykładnik (z) = wykładnik (x).

● **Konwersja na liczbę całkowitą.** Jeśli wykładnik jest większy od liczby bitów części ułamkowej, to konwersja musi być bezbłędna. Jeśli nie — to liczba musi być znormalizowana.

● **Konwersja nieznormalizowanej liczby na liczbę dziesiętną o postaci $M\times10^N$,** gdzie M i N są liczbami całkowitymi. W reprezentacji liczby M winna być odpowiednia liczba zer nieznaczących w celu wskazania stopnia nieznormalizowania.

● **Porównanie.** Wynik porównania jest taki, jakby obydwa argumenty były najpierw znormalizowane.

Ponadto, powinien być przewidziany tryb operacji normalizujący wszystkie argumenty przed wykonaniem jakichkolwiek działań. Jest to usilnie zalecane przez autorów normy.

WYJĄTKI

Domyślnym sposobem reakcji na pojawienie się wyjątku jest kontynuacja programu bez przerwania programowego (ang. trap). Powinna istnieć jednak możliwość obsługi przerwań programowych przez użytkownika.

Jeśli przerwanie programowe nie pojawia się, a zachodzi właśnie sytuacja wyjątkowa, to winien być ustawiony specjalny wskaźnik, który może być skasowany tylko na polecenie użytkownika. Użytkownik musi mieć możliwość zmiany wskaźnika wyjątku, sprawdzania go oraz zapamiętywania wszystkich wskaźników jednocześnie. Istnieje pięć rodzajów wyjątków.

1. Nielegalna operacja.

Ten rodzaj wyjątku dzieli się na dwie klasy: nielegalny argument i nielegalny wynik.

● **Nielegalny argument.** Sytuacja taka występuje w przypadku:
— operacji, w której jeden z argumentów jest nie-liczbą wywołującą przerwanie, a przerwanie jest zabronione
— dodawania lub odejmowania $\infty\pm\infty$ w arytmetyce rzutowej oraz odejmowania $(+\infty)+(-\infty)$ w arytmetyce afinicznej
— mnożenia 0 przez ∞
— dzielenia 0/0, ∞/∞; lub gdy dzielnik jest nieznormalizowany, a dzielna jest skończona i niezerowa
— x REM y, gdy y jest zerem lub jest nieznormalizowane bądź gdy x jest nieskończone
— obliczania pierwiastka kwadratowego z niezerowej liczby ujemnej, z nieskończoności w arytmetyce rzutowej lub z liczby nieznormalizowanej
— konwersji liczby zmiennoprzecinkowej na całkowitą lub dziesiętną, powodującej powstanie przepełnienia, wartości nieskończonej lub nie-liczby w danym formacie wynikowym, a nie może to być zasygnalizowane inaczej — porównania przy użyciu relacji $<,\leqslant,\geqslant,>$ lub ich negacji, gdy relacja jest nieokreślona (ang. unordered).

● **Nielegalny wynik.** Wyjątek taki pojawia się, gdy wynik operacji jest nieunormowany, ale znormalizowany. Jeśli jednocześnie wystąpią wyjątki przepełnienia, niedokładnego wyniku lub niedomiaru, to pierwszy musi zostać zgłoszony wyjątek niedokładnego wyniku.

2. Dzielenie przez zero. Wynikiem takiej operacji (gdy dzielna jest skończona i niezerowa) winna być ∞ z właściwym znakiem.

3. Nadmiar (przepełnienie).

Wyjątek taki winien być zasygnalizowany, gdy zaokrąglony wynik jest skończony, lecz jego wykładnik jest zbyt duży dla formatu docelowego. W przypadku, gdy zaokrąglamy w kierunku $-\infty$ lub $+\infty$, nie sygnalizujemy przepełnienia, a wynik musi być zaokrąglony według poniższych reguł.

● **Zaokrąglanie w kierunku −∞.** Liczby znormalizowane (ang. normalized) dodatnie są przedstawiane jako największe liczby danego formatu, zaś liczby nieunormowane dodatnie — przy użyciu najwyższego wykładnika w danym formacie bez zmiany mantysy.

● **Zaokrąglanie w kierunku +∞.** Nadmiar ujemny znormalizowany prowadzi do najmniejszej liczby ujemnej, zaś nadmiar ujemny nieunormowany — do liczby z największym wykładnikiem dla danego formatu bez zmiany mantysy.

W obu powyższych przypadkach niewłaściwe wyniki mogą powstać tylko w czasie konwersji z formatu rozszerzonego na format zasadniczy. W pozostałych przypadkach, jeśli nadmiar nie powoduje przerwania programowego, wynikiem musi być ∞ z odpowiednim znakiem.

4. Niedomiar.

Wyjątek tego rodzaju pojawia się wtedy, gdy:
● wynik nie jest zerem, lecz jego wykładnik jest zbyt mały, aby mógł być przedstawiony w formacie wynikowym bez odnormalizowania
● w formacie rozszerzonym iloczyn lub iloraz niezerowych argumentów po zaokrągleniu nie różni się od zera.
Jeśli niedomiar nie wywołuje przerwania programowego, to wynik musi być najpierw odnormalizowany, a następnie zaokrąglony. Ponadto, musi być ustawiony wskaźnik niedomiaru, chyba że zaokrąglanie odbywa się w kierunku $+\infty$ lub $-\infty$. Jeśli niedomiar wywołuje przerwanie programowe, to wynik musi być przekazany do procedury obsługi.

5. Wynik niedokładny.

Sytuacja taka musi być sygnalizowana w przypadku, gdy nie pojawi się zgłoszenie operacji nielegalnej, a wynik nie jest dokładny lub — gdy wynik jest nadmiarem, który nie jest sygnalizowany przerwaniem programowym. Otrzymany wynik musi mieć postać zaokrągloną lub nadmiarową.

PRZERWANIA PROGRAMOWE

Użytkownik winien mieć możliwość zażądania, aby w każdej z opisanych sytuacji wyjątkowych pojawiło się przerwanie programowe lub też — aby takie przerwanie było zabronione (ang. disabled). Jeśli przerwanie jest zabronione, to odpowiedni wyjątek musi być traktowany w sposób opisany w poprzednim punkcie. Jeśli pojawi się wyjątek, dla którego przerwanie programowe jest dozwolone (ang. enabled), wtedy wykonanie programu musi być zawieszone, a wynik — jeśli go określono w poprzednim punkcie — musi zostać przekazany do procedury obsługi przerwania programowego (ang. trap handler).

Użytkownik musi mieć możliwość określenia procedury obsługi przerwania programowego (to znaczy — wyboru albo zaprogramowania takiej procedury). Musi ona mieć postać procedury udostępniającej liczbę, która będzie wstawiona zamiast niewłaściwego wyniku. Wynik ten pozostaje nieokreślony do czasu zakończenia procedury obsługi przerwania. Podobnie, wskaźnik wyjątku wywołującego przerwanie programowe może być nieokreślony do czasu ustawienia go (lub skasowania) przez procedurę obsługi. Gdy nastąpi przerwanie programowe, to procedura ta musi mieć możliwość określenia: rodzaju wyjątku, rodzaju operacji, formatu końcowego, wartości argumentów w przypadku pojawienia się nielegalnych argumentów lub dzielenia przez zero, a także — poprawnie zaokrąglonego wyniku wraz z informacją nie mieszczącą się w formacie

końcowym w przypadku nadmiaru, niedomiaru, niedokładnego lub nielegalnego wyniku.

ZALECANE FUNKCJE I PREDYKATY

Norma zaleca wprowadzenie następujących funkcji i predykatów:

- copysign (x,y) — udostępnia x ze znakiem y; abs(x)=copysign(x,1.0)
- —x — x z przeciwnym znakiem
- scalb(x,N) — udostępnia iloczyn x przez 2^N dla całkowitych N; polega to na dodaniu N do wartości wykładnika i sprawdzeniu wyjątków
- logb(x) — udostępnia wartość wykładnika x (nieprzesuniętego) w postaci liczby całkowitej ze znakiem w formacie takim jak x, z wyjątkiem sytuacji, gdy: logb(0)=—∞, logb(∞)=+∞ i logb (nie-liczba) = ta_sama_nie-liczba; jeżeli x jest dodatnie i skończone, to $1 \leqslant scalb(x,-logb(x)) < 2$, z wyjątkiem sytuacji, gdy x jest nieunormowane lub nieznormalizowane
- nextafter(x,y) — udostępnia następną po x liczbę w kierunku y; jeśli x=y lub jeśli x lub y jest równe ∞ w arytmetyce rzutowej, lub jeśli x lub y są nie-liczbami, to wartością tej funkcji ma być x
- finite(x) — udostępnia wartość TRUE, gdy $-\infty \leqslant x \leqslant +\infty$, w przeciwnym przypadku — wartość FALSE
- isnan(x) — udostępnia wartość TRUE, gdy x jest nie-liczbą (równoważna operacji x≠x)
- x<>y — udostępnia wartość TRUE, gdy x<y lub x>y; różni się ona od x≠y, która oznacza NOT(x=y); jest zawsze operacją legalną
- unordered(x,y) — udostępnia wartość TRUE, gdy x jest nieokreślone względem y; jest zawsze operacją legalną.

LITERATURA

[1] Stevenson D. (ed.): A Proposed Standard for Binary Floating-Point Arithmetic. Computer, Vol. 14, No. 4, 1981

[2] Fateman R. J.: High-Level Language Implications of the Proposed IEEE Floating-Point Standard. ACM Trans. on Programming Languages and Systems, Vol. 4, No. 2, 1982.

II/B. Reprezentacja, przepływ i przekształcanie danych

Samotesty, opracowane i opublikowane przez Association for Computing Machinery, nie stanowią standardu egzaminacyjnego i są przeznaczone wyłącznie do samodzielnego sprawdzania własnej wiedzy.

PYTANIA

11. Taśmy magnetycznej z wieloma tysiącami zblokowanych rekordów na ogół nie można bezpośrednio wykorzystywać na komputerach o odmiennej architekturze, ponieważ...

    a) — taśmy magnetyczne mogą być czytane tylko przez komputer, który je zapisał
    b) — powstaną błędy parzystości
    c) mogą zachodzić różnice w formatach danych
    d) rekordów z danymi nie wolno łączyć w bloki

12. W niektórych rozkazach można wykorzystywać ich pola adresowe do bezpośredniego zapisu danych. Zaletą takiego wykorzystania pola adresowego rozkazu jest...
    a) umożliwienie adresowania pośredniego (ang. indirect addressing)
    b) zmniejszenie liczby cykli odczytu
    c) możność bezpośredniego adresowania dowolnych pól danych
    d) uproszczenie procesu wnoszenia zmian w programie

13. Załóżmy, że procesor ma dwa rejestry (R1 oraz R2) zawierające dane binarne, wykorzystywane jako argumenty w rozkazie:

    EOR Ri, Rj

    który tworzy z odpowiadających sobie bitów zawartości tych rejestrów wartość logiczną ALBO (Exclusive OR) pozostawianą w Ri. Załóżmy, że wykonano kolejno trzy rozkazy:

    EOR R1, R2
    EOR R2, R1
    EOR R7, R2

    — co otrzymamy w wyniku?

    a) rejestry R1 oraz R2 zostaną wyzerowane (000...0)
    b) rejestry R1 oraz R2 otrzymają wartość binarną (111...1)
    c) rejestr R1 będzie zawierać arytmetyczną sumę oryginalnych zawartości R1 oraz R2
    d) zawartości R1 oraz R2 zostaną zamienione miejscami

14. Wymień wszystkie cztery parametry, jakie musi mieć podane procesor wejścia-wyjścia (np. kanał) dla transmisji bloku danych między pamięciami operacyjną i zewnętrzną.

15. Podaj trzy kroki, jakie na poziomie architektury wewnętrznej są niezbędne przy odczytywaniu pojedynczego rekordu w urządzeniu pamięciowym z dysku z ruchomym mechanizmem głowic.

16. Określ liczby całkowite (dziesiętne) odpowiadające czwórce bitów 1010 w każdej z wymienionych interpretacji arytmetycznych:

    a) oznakowany moduł (ang. signed magnitude): ...
    b) uzupełnienie do 2 (ang. two's complement): ....
    c) uzupełnienie do 1 (ang. one's complement): ....
    d) kod ośmio-nadmiarowy (ang. excess-8): ....

17. Które z poniższych urządzeń najbardziej rażąco nie nadaje się, jako konwerter reprezentacji zewnętrznej danych np. z kart dziurkowanych na reprezentację wewnętrzną (np. na 8-bitowy kod alfanumeryczny)?
    a) jednostka sterująca (ang. control unit)
    b) procesor główny (CPU)
    c) szyna danych (ang. data bus)
    d) kanał zewnętrzny (ang. channel)

18. Wskaż co najmniej trzy bardzo istotne różnice między kodami BCD, EBCDIC i ASCII — tj. bezpośrednim kodowaniem dwójkowym cyfr dziesiętnych (ang. Binary Coded Decimal), kodowaniem bezpośrednim cyfr dwójkowych z rozszerzeniem na symbole nienumeryczne (ang. Extended Binary Coded Decimal Interchange Code) oraz znormalizowanego amerykańskiego kodu do wymiany informacji (ang. American Standard Code for Information Interchange).

19. Załóżmy, że nieoznakowane arytmetycznie mantysy liczb zmiennoprzecinkowych zajmują w pamięci 36 bitów. Niechaj d oznacza maksymalną liczbę cyfr znaczących równoważnego zapisu dziesiętnego. Które równanie jest właściwie do wyznaczenia wartości d?
    a) $36 = 10 \cdot d$
    b) $36 \cdot 2 = 10 \cdot d$
    c) $36^2 = d^{10}$
    d) $10^d = 2^{36}$

20. Do przedstawiania liczb binarnych (dodatnich i ujemnych) stosuje się zapis:

    a) znak moduł
    b) uzupełnień do 1
    c) uzupełnień do 2

    Powiąż każdą z opisanych niżej cech z odpowiednimi zapisami, dla których jest ona prawdziwa:

    A) wystąpują dwie różne postacie liczby 0
    B) przedział liczb dodatnich i ujemnych jest identyczny (liczba o znaku przeciwnym należy też do reprezentacji maszynowej)
    C) przy dodawaniu dwóch liczb różnoznakowych, może pojawić się przeniesienie cykliczne
    D) układ bitów przypomina naturalny sposób zapisywania liczb dodatnich i ujemnych

ROZWIĄZANIA:

11. c; 12. b; 13. d; 14. Adres obszaru w pamięci operacyjnej, adres obszaru w pamięci zewnętrznej, liczność danych, kierunek przesyłania np. READ/WRITE; 15. Ustawienie mechanizmu dostępu (np. SEEK), zlokalizowanie żądanego rekordu (np. SEARCH), odczytanie tego rekordu (np. READ); 16. a) —2, b) —6, c) —5, d) —2; 17. c; 18. Kolejność dobierania (ang. collating), długość binarna kodu pojedynczego znaku, zakres rozróżnialnego alfabetu; 19. d; 20. a) A, B, D, b) A, B, C.

LITERATURA

ad 11. Gear W.; Computer Organization and Programming. II ed., McGraw-Hill, pp. 20, 50—52.

ad 12. Wegner P.: Języki programowania, struktury informacji i organizacja maszyny cyfrowej, PWN, 1979

ad 13. Gear, ibid. p. 290

ad 14. Gear, ibid. p. 253
Wegner, ibid.

ad 15. Gear, ibid. p. 233
Wegner, ibid.

ad 16. Gear, ibid. pp. 19. 52—58
Blaauw G.: Digital System Implementation, Prentice-Hall, pp. 237—342
Flores I.: Arytmetyka maszyn cyfrowych. WNT, 1970, ss. 32—36

ad 17. Chapin N.: An Introduction to Automatic Computers II ed. Van Nostrand, p. 67

ad 18. Gear, ibid. pp. 20—21

ad 19. Gear, ibid. pp. 50—52

ad 20. Blaauw, ibid. pp. 333—342
Gear, ibid. pp. 18—20, 52—58
Stone H.: Introduction to Computer Architecture, Science Research Associates, pp. 34—44

Adaptowali z angielskiego:
ADAM B. EMPACHER
LUDWIK J. ROSSOWSKI

Zjazd PTI

12 maja 1984, po trzech latach istnienia, Polskie Towarzystwo Informatyczne zwołało I Walny Zjazd. Uczestniczyło w nim 132 członków zwyczajnych, przedstawiciele członków wspierających oraz zaproszeni goście. Po podsumowaniu pierwszego okresu działalności Towarzystwa (od Zjazdu Założycielskiego), wybrano nowe jego władze.

Prezydium Zarządu Głównego:

prezes — Władysław M. Turski

wiceprezesi — Wojciech Cellary, Jacek Irlik, Stanisław Jaskólski
skarbnik — Marek Maniecki
członkowie — Jacek Bańkowski, Andrzej Blikle, Stanisław Lepetow, Mieczysław Muraszkiewicz, Stanisław Waligórski

Sekretarz generalny PTI — Andrzej Wiśniewski.

Do ważniejszych ustaleń Zjazdu można zaliczyć — przyjęcie regulaminu nadawania przez PTI stopni specjalizacyjnych z informatyki oraz zapowiedź zorganizowania ogólnopolskiego konkursu im. Jerzego Trybulskiego na najlepsze prace z dziedziny zastosowań informatyki.

Główną uchwałę Zjazdu publikujemy poniżej. Jest to rzeczowy głos w sprawie polskiej informatyki, pod którym nasza redakcja może się podpisać bez zastrzeżeń. Nie mogliśmy przedstawić jej wcześniej, bowiem tekst otrzymaliśmy dopiero po opublikowaniu go na łamach BIULETYNU PTI nr 5—6/1984 — we wrześniu ub. r.

Z.G.

Uchwała I Walnego Zjazdu PTI w sprawie rozwoju informatyki w Polsce

Uczestnicy I Walnego Zjazdu Polskiego Towarzystwa Informatycznego — po zapoznaniu się z referatem sprawozdawczym ustępującego Zarządu Głównego PTI i po wysłuchaniu głosów w dyskusji — zajęli następujące stanowisko w sprawie rozwoju informatyki w Polsce.

1. Głównym warunkiem pomyślnego rozwoju informatyki w Polsce jest racjonalny stosunek władz do informatyki i jej zastosowań we wszystkich dziedzinach życia gospodarczego i społecznego. Twierdzimy, że w dotychczasowym stosunku władz państwowych i lokalnych, gospodarczych i administracyjnych do informatyki i jej zastosowań dominuje niekonsekwencja lub brak zrozumienia tego, czego można spodziewać się po informatyce i czego można od niej żądać. Prawidłowe zastosowania informatyki są na tyle ważne dla rozwoju gospodarczego państwa i rozwoju społecznego jego obywateli, a zaniedbania w tej dziedzinie tak mocno zagrażają utrzymaniu Polski w gronie państw rozwiniętych, że informatyka powinna znaleźć się pod szczególną opieką państwa.

2. Uważamy, że jedynym rozsądnym podejściem do kompleksu problemów związanych z informatyką jest uznanie prymatu jej zastosowań. Polityka państwa wobec informatyki powinna przeto wywodzić się z jasno określonego planu rozwoju jej zastosowań, a inne poczynania, takie jak planowanie produkcji urządzeń informatyki, finansowanie badań, polityka podatkowa, celna i płacowa dotyczące informatyki, powinny być pochodnymi planu rozwoju jej zastosowań. Uważamy, że formułować politykę państwa w tej mierze oraz skutecznie czuwać nad jej realizacją może tylko wyodrębniony organ centralny — rządowa agencja informatyki, niezależna od wpływów przemysłu oraz innych potężnych sił nacisku i wspomagana w tym działaniu przez radę złożoną z informatyków cieszących się zaufaniem całego środowiska.

3. Społeczeństwo polskie jest niedoinformowane co do rzeczywistych możliwości informatyki i korzyści płynących z jej powszechnego zastosowania. Zbyt często nieudolność administracji i kierownictwa gospodarczego różnych instytucji jest maskowana bałamutnymi stwierdzeniami o pomyłkach komputerów. Uważamy, że należy znacznie lepiej niż dotychczas uświadamiać społeczeństwu realia zastosowań informatyki. Sądzimy, że nakładanie formalnych samoograniczeń importowych na sprzęt informatyczny jest wysoce szkodliwe dla krajowej informatyki i gospodarki. Sprzęt informatyczny powinien być zwolniony z wszelkich opłat celnych niezależnie od jego rodzaju, a sam jego import nie powinien wymagać żadnych specjalnych zezwoleń administracyjnych. Jednymi regulatorami tego importu powinny być mechanizmy ekonomiczne.

4. Istotnym powodem małej efektywności zastosowań informatyki w Polsce jest krańcowy niedorozwój usług informatycznych zarówno w sensie wytwórstwa oprogramowania, jak i serwisu oraz doradztwa. Dlatego, naszym zdaniem, należy znieść wszelkie ograniczenia administracyjne wobec tworzenia państwowych, spółdzielczych i prywatnych firm, które mogłyby takie usługi świadczyć w sposób rzetelny. Jedynym kryterium istotnym przy udzielaniu licencji na prowadzenie działalności usługowej w tym zakresie powinny być kwalifikacje fachowe.

5. Dotychczasowe sposoby wiązania działalności produkcyjnej przedsiębiorstw wytwarzających sprzęt informatyczny z potrzebami zastosowań są całkowicie niewłaściwe. Jeżeli przemysł informatyczny ma się kierować wskazaniami czysto ekonomicznymi — a stanowisko takie nie jest pozbawione sensu — to należy umożliwić kra-

jowym przedsięwzięciom informatycznym ekonomiczne oddziaływanie na ów przemysł, a także udostępniać środki budżetowe na interwencyjne zakupy sprzętu zagranicznego, które konkurując mogłyby pobudzić poprawę jakości sprzętu produkcji krajowej. Należy przy tym wziąć pod uwagę fakt, że — ze względu na istniejącą strukturę cen i sposób finansowania — wiele bardzo ważnych zastosowań informatyki nie może być samofinansowanych. Wynika stąd konieczność uzupełniającego finansowania budżetowego za pośrednictwem rządowej agencji informatyki. Jednocześnie występujemy przeciwko bezpośredniemu dotowaniu przemysłu komputerowego. Środki budżetowe dla tego przemysłu muszą przechodzić przez instytucje żywotnie zainteresowane rozwojem zastosowań informatyki na swoim obszarze.

6. Uważamy za konieczne utrzymanie wysokiego poziomu badań podstawowych w dziedzinie informatyki, dzięki któremu jest ciągle jeszcze utrzymywany bezpośredni kontakt z rozwojem tej dziedziny na świecie. Wzorem innych, lepiej przewidujących państw, nakłady na badania należy zwiększać szybciej niż w innych obszarach nauki i techniki, tutaj bowiem odbywa się światowa rywalizacja decydująca o przyszłym poziomie całej nauki i techniki w kraju.

7. Należy rozwijać kształcenie informatyków na wyższych uczelniach oraz zwiększać nasycenie technikami informatycznymi wszystkich programów dydaktycznych szkolnictwa wyższego. Całe szkolnictwo wyższe powinno być wyposażone w nowoczesne urządzenia informatyczne, i to wyprzedzająco w stosunku do innych obszarów życia gospodarczego i społecznego, gdyż tylko to pozwoli złagodzić ciągły deficyt odpowiednio przygotowanych kadr informatyków.

8. Należy dążyć do stopniowego wprowadzania informatyki do szkół średnich, stwarzając im możliwości korzystania ze środków informatyki (sprzęt i oprogramowanie) w całym procesie nauczania. Pozbawione są sensu przedsięwzięcia zmierzające do wprowadzania informatyki do szkolnictwa średniego i podstawowego bez zagwarantowania dostatecznej ilości niezawodnego i dobrze oprogramowanego sprzętu. Werbalne nauczanie informatyki jest bezowocne, a nawet szkodliwe, jak wszelka działalność fikcyjna. Uważamy, że przeróżne kursy dokształcające w dziedzinie informatyki, zwłaszcza zaś kursy dla osób bez uprzedniego przygotowania fachowego w tej dziedzinie, mają sens tylko wówczas, gdy łączą się z poważnie prowadzonymi zajęciami praktycznymi.

9. Uważamy, że praca informatyków nie jest należycie honorowana finansowo i prestiżowo. Układy zbiorowe, siatki płac itp. instrumenty określające uposażenie informatyków nie odzwierciedlają ani wkładu pracy potrzebnego na zdobycie i utrzymanie kwalifikacji, ani wartości pracy wykwalifikowanego informatyka na światowych rynkach pracy. Sytuacja ta prowadzi do szkodliwych frustracji, porzucania zawodu i dość znacznej emigracji zarobkowej. Zwracamy uwagę, że aczkolwiek emigracji zarobkowej można dość łatwo położyć kres w drodze odpowiednich decyzji administracyjnych, to jednak bez istotnej poprawy warunków płacowych i bez stworzenia perspektyw rozwoju informatyki nie da się zahamować porzucania zawodu przez informatyków. Sądzimy, że wszelkie społecznie i gospodarczo uzasadnione przejawy i zamiary zastosowań informatyki powinny być popierane zarówno przez państwo, jak i przez społeczeństwo. Wyrazami tego poparcia powinny być odpowiednie postępowania podatkowe, kredytowe i celne z jednej strony, a działalność stowarzyszeń zawodowo-społecznych i zrzeszeń hobbystów — z drugiej strony.

I Walny Zjazd Polskiego Towarzystwa Informatycznego zobowiązuje Zarząd Główny PTI do intensywnych działań na rzecz realizacji postulatów niniejszej uchwały.

KONKURS

Historia zakładu przemysłowego bądź wielkiego przedsięwzięcia inżynieryjnego, opracowana w latach 1944—1984

Polskie Towarzystwo Historii Techniki, Naczelna Organizacja Techniczna i Redakcja tygodnika „Polityka" ogłosiły pierwszy ogólnokrajowy konkurs otwarty, obejmujący historię zakładów przemysłowych w Polsce.

Regulamin Konkursu został zamieszczony w „Polityce" i „Przeglądzie Technicznym".

Mogą w nim brać udział prace publikowane i nie publikowane, powielane i w maszynopisie — opracowane lub wydane w 40-leciu powojennym.

Organizatorzy proszą o nadsyłanie prac do 15 kwietnia 1985 r. pod adresem:

Polskie Towarzystwo Historii Techniki — Muzeum Techniki NOT Pałac Kultury i Nauki, 00-901 Warszawa

Wynik z komputera

We wrześniu ub.r. w wodach jeziora Ukiel w Olsztynie odbywała się w licznej obsadzie międzynarodowej piąta edycja Pucharu Europy w Podwodnej Orientacji — obsługiwana przez nowy polski mikrokomputer IMZ-80, produkowany przez PPZ IMPOL II w Warszawie.

W czerwcu ub.r. ZETO Olsztyn zakupiło jeden z pierwszych egzemplarzy IMZ-80. Bazuje on na mikroprocesorze Z80, wyposażony jest w 64 K pamięci operacyjnej, klawiaturę i monitor ekranowy. Pamięć zewnętrzną stanowi magnetofon kasetowy (programy oraz zbiory danych), a uzupełnienie aktualnej konfiguracji — drukarka znakowo-mozaikowa DZM-180.

Oprogramowanie zapewniające obsługę zawodów wykonałem w języku BASIC. Obejmowało ono cztery wielofunkcyjne programy odpowiadające poszczególnym konkurencjom, a w ich ramach m.in.:

- zakładanie listy startowej oraz jej aktualizację
- wprowadzanie rezultatów uzyskiwanych przez poszczególnych zawodników lub drużyny
- wyprowadzanie informacji o aktualnej sytuacji w konkurencji
- odtwarzanie stanu konkurencji w dowolnym momencie jej trwania
- wydruk ostatecznych rezultatów.

Zasadniczą trudność stanowiło opracowanie niezawodnych i efektywnych algorytmów naliczania punktów i ustalania kolejności. Przykładowo — w konkurencji drużynowej mężczyzn na ostateczny wynik składała się suma punktów uzyskanych na trasie za dziesięć zaliczonych boi oraz punktów za uzyskany na mecie czas i odchylenie od punktu centralnego mety (wg kilku różnych tabel). Przy równej liczbie punktów, o kolejności decydował czas z dokładnością do dziesiątych części sekundy.

Olsztyńska impreza była dla mikrokomputera IMZ-80 testem niezawodności (wielogodzinna praca non-stop), a dodatkowo — miała pokazać, że możliwa jest obsługa zawodów sportowych na sprzęcie mikroinformatycznym polskiej produkcji oraz że nie trzeba tworzyć do tych celów wieloosobowego sekretariatu wyposażonego w stosy dokumentów, kalkulatory, maszyny do pisania, kserografy...

Obsługę zawodów na mikrokomputerze zapowiedziano ekipom na konferencji technicznej. Na wielu twarzach zauważyłem cień niedowierzania, a jednocześnie — zaciekawienia. Do tej pory dla obsługi europejskich zawodów w podwodnej orientacji mikrokomputery były stosowane rzadko — tylko do naliczania rezultatów. A i to nie zawsze odbywało się bez kłopotów, nawet w krajach bardziej zaawansowanych w komputeryzacji.

Mikrokomputer IMZ-80 zainstalowany został na plaży miejskiej w drewnianym, „chwiejącym się" pawiloniku. Mimo że losowanie zakończyło się tuż przed północą, zawodników i kibiców przybywających od ósmej rano nad jezioro powitały wywieszone (i wręczane) listy startowe. Karta startowa każdego zawodnika była dostarczana bezpośrednio po ukończeniu przez niego konkurencji. Zdarzało się, że zawodnik jeszcze przed wyjściem na brzeg słyszał ogłaszany przez spikera komunikat o uzyskanym wyniku oraz aktualnej pozycji.

Po pewnym czasie w niewielkim pomieszczeniu, w którym pracował IMZ-80, robiło się chwilami ciasno. Kierownicy drużyn, zawodnicy i — oczywiście — kibice chcieli zobaczyć pracę sekretariatu „od kuchni". Dziennikarze byli wyraźnie zadowoleni z otrzymywania kompletnych wyników w kilka minut po zakończeniu konkurencji. Miło było słyszeć opinie, że takiego tempa jeszcze nie widzieli. Spore zainteresowanie systemem wykazywali też — ku memu zaskoczeniu — przedstawiciele ekip zagranicznych.

Sprzęt i oprogramowanie, w opinii wszystkich obserwatorów, spisały się bez zarzutu. A — co ważne — dla wielu osób było to pierwsze bezpośrednie zetknięcie z użytkową mikroinformatyką. Teraz zapewne także organizatorzy innych imprez sportowych skorzystają z podobnej oferty.

JERZY KUROWSKI
ZETO OLSZTYN

Robotnicza Spółdzielnia „Prasa-Książka-Ruch" Centrala Kolportażu Prasy i Wydawnictw sprzeda:

karty analityczne 90-kolumnowe do maszyn analitycznych firmy ARITMA w liczbie 2 164 000 sztuk po 6 groszy sztuka

Informacji o sprzedaży udziela Dział Zaopatrzenia tel. 20-48-71, ul. Towarowa 28, 00-958 Warszawa, skr. poczt. 12.

EO/139/K/85

Świat superkomputerów CRAY

Wielu problemów nauki i techniki nie rozwiązano do dziś z powodu braku możliwości uzyskania wyników w „rozsądnym" czasie. Dalszy więc postęp w różnorodnych dziedzinach nauki oraz rozwój przemysłowy jest uzależniony od wzrostu szybkości obliczeń w komputerach.

Od czasu pierwszych komputerów, ENIACA, MANIACA i innych, postęp wyrażający się wzrostem mocy przetwarzania danych można określić mianem eksplozji. Zastąpienie lamp tranzystorami, a tych układami scalonymi o coraz większej skali integracji doprowadziło do zmniejszenia czasu przełączania układów logicznych, a więc znacznego zwiększenia szybkości działania. Miniaturyzacja sprzętu umożliwiła też zamknięcie wielkich dotychczas maszyn — w walizkę. Komputery osobiste oraz superminikomputery zrewolucjonizowały technikę przetwarzania danych.

Jednak dla wielu zastosowań i tego było za mało, a przede wszystkim — sprzęt ten był zbyt wolny. Z kolei trudno jest ze względów technologicznych przekroczyć bariery nanosekundowego cyklu działania procesora. Począwszy od lat sześćdziesiątych, rozpoczęto prace nad komputerami o dużej szybkości — przez zwielokrotnienie fragmentów lub też całych procesorów. Zaprojektowano komputery potokowe, wektorowe, macierzowe, asocjacyjne o architekturach daleko odbiegających od klasycznej. Komputery te, ze względu na posiadane moce przetwarzania, zostały ochrzczone mianem s u p e r k o m p u t e r ó w.

SUPERKOMPUTER CRAY

Jednym z projektów, z powodzeniem wdrożonych do produkcji, stał się projekt superkomputera CRAY-1, którego ojcem był Mr Cray, były główny projektant znanej firmy CONTROL DATA CORPORATION. Podstawowym założeniem w zaprojektowanej architekturze komputera było uzyskanie możliwie maksymalnej szybkości przetwarzania, a więc:

- dla zmniejszenia czasu przekazywania sygnałów — uzyskano skrócenie połączeń, rozmieszczając elementy sprzętu w pudle będącym walcem (następca CRAYA będzie kulą?!); otrzymano czas cyklu równy 12,5 ns
- dla przyspieszenia obliczeń — wszystkie jednostki arytmetycznt realizujące pojedyncze operacje zmiennoprzecinkowe wykonano jako jednostki potokowe; cechą jednostki potokowej jest możliwość rozpoczęcia wykonywania operacji dla nowych argumentów, przed pełnym zakończeniem wykonania tej operacji dla aktualnych argumentów
- maksymalne zatrudnienie dla jednostek arytmetyczno-logicznych uzyskano lokalizując wszystkie argumenty operacji w rejestrach skalarnych lub w zbiorze rejestrów tworzących wektor argumentów
- aby nie oczekiwać na pobranie kolejnego rozkazu, sekwencje rozkazów umieszczone są również w tablicy rejestrów rozkazów
- dostarczenie takiej porcji informacji do rejestrów wymagało zastosowania dużej pamięci półprzewodnikowej (do 4 mln słów 64-bitowych) o specjalnym wielokrotnym równoczesnym dostępie.

Dzięki tym posunięciom, uzyskano maksymalną szybkość pracy do 250 mln operacji na sekundę, przy czym osiągnięcie tej szybkości jest uzależnione od występowania równoległości w dostarczonym do obliczeń algorytmie.

Dla porównania — rozwiązanie nieliniowych równań opływu powietrza wokół skrzydła samolotu zajęło 18 godzin pracy komputera IBM 360, a mniej niż 1 minutę pracy superkomputera CRAY-1.

Procesor superkomputera współpracuje z trójpoziomową pamięcią zewnętrzną. Pierwszy poziom zlokalizowany na dyskach przechowuje pliki o pojemności 38 mld bitów i średnim czasie dostępu 35 ms. Kolejny poziom jest zautomatyzowanym zbiorem pamięci taśmowych o pojemności ok. 500 mld bitów i czasie dostępu 20 s. Ostatni poziom tworzy biblioteka złożona ze zbioru 2000 taśm, przechowująca pliki o pojemności 2 trylionów bitów danych. Organizacja pamięci zewnętrznej może być dopasowywana do potrzeb zależnych od przeznaczenia superkomputera.

Superkomputer CRAY-1 ma już następców: dwuprocesorowy CRAY-1S oraz szybszy CRAY X-MP o czasie cyklu równym 9,5 ns.

PRZEZNACZENIE SUPERKOMPUTERÓW

Zastosowania superkomputerów obejmują dziedziny z zakresu podstawowych badań naukowych oraz praktycznych zastosowań w przemyśle. Przykładowymi obszarami użycia superkomputerów są:

— banki informacji z bibliotekami programów obliczeniowych
— badania naukowe w fizyce, chemii i astronomii
— prognozowanie pogody
— analiza wyników poszukiwań geologicznych ropy oraz metod jej przechowywania
— symulacje całościowych zachowań projektów samolotów
— symulacje badań tunelowych nowych projektów karoserii samochodowych
— projekty rozmieszczeń układów elektronicznych dwu- i trójwymiarowych
— modelowanie systemów bezpieczeństwa reaktorów atomowych dla energetyki jądrowej

— wytwarzanie i przetwarzanie obrazów graficznych.

Poniżej zostaną zaprezentowane sposoby i efekty zastosowań superkomputerów w wybranych dziedzinach.

Bank informacji

Przykładem banku informacji jest sieć pracująca w USA na potrzeby programu badań w dziedzinie kontrolowanej syntezy termojądrowej. Narodowe Centrum Komputerowe Magnetycznej Fuzji Jąder zostało utworzone przez Departament Energii USA w 1974 roku. Centrum to wyposażone jest w superkomputery CRAY-1, CRAY-1S oraz w starszą maszynę CDC 7600.

Użytkowanie sieci komputerowej rozpoczęto w roku 1976. Obecnie z usług Centrum korzysta ok. 2000 badaczy w 23 ośrodkach. Sieć umożliwia wymianę informacji, danych oraz wykorzystywanie umiejętności ludzi zatrudnionych w innych ośrodkach, wszyscy też mogą korzystać z możliwości obliczeniowych komputera centralnego. Centrum udostępnia bazę danych LIBRIS, w której przechowywane są abstrakty prac dotyczących badanego problemu. Każdy z użytkowników może wprowadzać do tej bazy abstrakty własnych prac i przez podanie swego kodu umożliwić kontakt innym użytkownikom sieci. Program DOCUMENT pozwala użytkownikom na studiowanie ok. 16 000 stron dokumentacji dotyczącej oprogramowania systemu. Dowolny dokument może być udostępniony w postaci wydruku lub mikrofilmu. Dla ułatwienia komunikacji z systemem, istnieje możliwość dialogu między komputerem centralnym, a prywatnymi minikomputerami użytkowników.

Wiadomości i poczta mogą być elektronicznie przetwarzane i przesyłane przez każdego użytkownika do pozostałych abonentów sieci. Odbiorca, otrzymawszy sygnał przybycia wiadomości, może zażądać wyświetlenia jej na monitorze lub wydrukowania, a następnie może podjąć decyzję o przechowaniu tej informacji lub jej zlikwidowaniu. System umożliwia utrwalanie obrazów na taśmie filmowej 35 mm, czarno-białej lub kolorowej.

Główne linie sieci to łącza satelitarne o prędkości przekazu 56 000 bitów/s. Linie główne połączone są z liniami telefonicznymi o prędkości przekazu 9600 bitów/s. Dodatkowa sieć komunikacyjna kontroluje przepływy wiadomości między komputerami. Centrum sieci połączone jest z lokalnymi stacjami obsługi użytkowników, wyposażonymi w komputery systemu DEC-NKL. Stacje lokalne gromadzą dane dotyczące przeprowadzanych na miejscu eksperymentów oraz umożliwiają wykonanie prostszych obliczeń.

Rozwiązywanie równania stanu jednoskładnikowej plazmy

Jednoskładnikowa plazma jest to układ jonów zanurzonych w jednorodnym morzu elektronów, przy czym całość jest elektrycznie obojętna. Rozwiązanie równania stanu takiej plazmy jest oparte na metodzie Monte-Carlo. Po ustaleniu gęstości i temperatury plazmy i obliczeniu gęstości takiego układu, wybiera się losowo jeden jon i poddaje losowemu zaburzeniu — ruchowi, po czym ponownie obliczana jest energia układu. Jeśli wartość energii okazuje się mniejsza niż poprzednio, to ruch jest akceptowany, w przeciwnym razie — ruch jest akceptowany z prawdopodobieństwem opisywanym przez rozkład Boltzmanna. Działania te są powtarzane dla następnych losowo wybieranych jonów. Punkt równowagi układu zostaje osiągnięty, gdy energia stabilizuje się. Teraz można wziąć pod uwagę nową gęstość i nową temperaturę i powtórzyć to samo postępowanie.

Superkomputer CRAY-1 rozwiązał to zadanie po siedmiu godzinach pracy. Otrzymano też znacznie dokładniejsze od dotychczas znanych oceny niektórych parametrów plazmy (różnice między starymi i nowymi wynikami sięgały 20%).

W rozwiązaniu zastosowano skuteczne algorytmy wektorowe. Efektywność działania tych algorytmów nie jest funkcją liczby wykonywanych operacji. Procesor wektorowy można traktować jako maszynę o dwóch stanach, która w jednym z tych stanów wykonuje działania arytmetyczne znacznie szybciej niż w drugim. Średni czas wykonywania operacji zależy więc od tego, jak operacje rozłożone są pomiędzy tymi stanami. Minimalizacja czasu pracy maszyny nie musi więc w tym przypadku oznaczać minimalizacji liczby wykonywanych operacji.

Podczas rozwiązywania równania plazmy metodą Monte-Carlo należało obliczyć potencjał oddziaływania między cząstkami, a to wymagało znajomości funkcji błędu, czyli całkowania po półnieskończonym przedziale. W komputerach pracujących sekwencyjnie tablicuje się funkcję błędu, a następnie w razie potrzeby interpoluje się wartości pomiędzy wartościami tablicowymi. Niestety, w maszynie CRAY-1 przeglądanie tablicy jest operacją powolną. Lepszą metodą okazała się aproksymacja wielomianowa. Metoda ta wymagała znacznie większej liczby operacji arytmetycznych, ale wykonywanych bardzo szybko i w sumie czas realizacji programu okazał się krótszy. Tak więc najlepszy algorytm w przypadku komputera pracującego wektorowo może być zupełnie inny niż dla komputera konwencjonalnego.

Kontrolowana synteza jąder

Postępy badań nad kontrolowaną reakcją termojądrową byłyby w ogóle niemożliwe bez pomocy superkomputerów. W szczególności niemożliwa byłaby analiza zachowania plazmy, gdy ściśnięta polem magnetycznym oddziałuje z cząstkami neutralnymi i źródłami pola elektromagnetycznego. Kompletny opis zachowania się plazmy wymagałby rozwiązania wielu złożonych równań ruchu płynów i cząstek. Analityczne rozwiązanie tych równań nie jest możliwe i dlatego w komputerach przeprowadza się zwykle badania symulacyjne. Jak twierdzą specjaliści, dopiero badania komputerowe pozwoliły teoretycznie zinterpretować wyniki doświadczeń, które przedtem były zupełnie niezrozumiałe.

Szczególnie trudne okazało się badanie wpływu zanieczyszczeń na zachowanie plazmy. Skomplikowany już opis plazmy jest tu dodatkowo utrudniony przez złożone obliczenia z zakresu fizyki atomowej. Na przykład — częściowo zjonizowany wolfram z pięćdziesięcioma związanymi elektronami ma skomplikowaną strukturę poziomów z tysiącami możliwych przejść. Badania komputerowe wykazały, że promieniowanie wolframu użytego w produkcji tokamaków będzie sześćdziesiąt razy większe niż wcześniej przewidywano. Potwierdziły to późniejsze eksperymenty. Kierując się tymi wynikami, zastąpiono wolfram innymi materiałami, co zmniejszyło straty promieniowania i pozwoliło osiągnąć wyższą temperaturę plazmy. Ważnym zadaniem przy syntezie termojądrowej jest także dobór kształtów, rozmiarów i położenia cewek oraz prądów przez nie płynących — w taki sposób, aby związanie plazmy polem magnetycznym było optymalne. W szczególności okazało się, że przy odpowiednim doborze cewek ten sam efekt można osiągnąć za pomocą prądów o połowę mniejszych.

Ciepło omowe jest zbyt małe, aby spowodować zapłon magnetycznie związanej plazmy. Dodatkowe podgrzewanie plazmy odbywa się za pomocą strumienia jonów. Komputer CRAY-1 umożliwił zbadanie skuteczności różnych konfiguracji elektrod, potencjałów przyspieszających i innych parametrów. Wynikiem był 30-procentowy wzrost skuteczności tej metody i oszczędności rzędu miliona dolarów.

Prognozowanie pogody

Dziesięć lat temu uważano, że wiarygodne dziesięciodniowe prognozy pogody są niemożliwe. Obecnie takie prognozy są systematycznie ogłaszane przez Europejskie Centrum Prognoz Średnioterminowych.

Należy podkreślić, że nowa generacja superkomputerów różni się w pewien istotny sposób od generacji poprzednich. Dawniej przejście do nowej generacji powodowane było przede wszystkim zmianami technologicznymi i większa szybkość obliczeń zapewniona była automatycznie bez udziału użytkownika. Inaczej dzieje się w przypadku superkomputera CRAY-1. Szybkość obliczeń zależy w dużym stopniu od tego, czy użytkownik potrafi przystosować sposób wykonywania obliczeń do architektury maszyny. Takie przystosowanie jest niezbędne w przypadku prognozowania pogody, które wymaga rozwiązywania wielu skomplikowanych równań różniczkowych i wykorzystywania ogromnej liczby danych. Optymalizacja algorytmów polega na tym, aby jak najwięcej procedur obliczeniowych wykonywanych było w sposób wektorowy. Takiej optymalizacji najtrudniej poddają się procedury przeglądania tablic, wykonywanie instrukcji warunkowych itp. W Europejskim Centrum

Prognoz Średnioterminowych wyniki takiej optymalizacji są następujące: czas zużywany przez centralny procesor na prognozowanie pogody w ciągu jednego dnia wynosi 160 s, gdy obliczenia przeprowadzane są skalarnie, oraz 76 s, gdy są przeprowadzane wektorowo.

W celu skonstruowania prognozy pogody dzieli się obszar Ziemi liniami współrzędnych na kwadraty i znajduje przybliżone rozwiązania równań dla tych kwadratów. Jeśli sieć współrzędnych jest gęstsza, to prognoza jest bardziej dokładna. Osiem lat temu wykonywano prognozy na trzy dni naprzód dla obszarów mniejszych niż półkula przy odległości linii współrzędnych rzędu 300—400 km. Począwszy od roku 1980, korzystając z komputera CRAY-1, codziennie ogłasza się prognozy na dziesięć dni naprzód. Dotyczą one całej Ziemi, a odległość linii współrzędnych wynosi 1,875 stopnia — zarówno w kierunku równoleżnikowym, jak i w południkowym. Wykonanie prognozy na jeden dzień trwa ok. 15—20 minut.

Filmy komputerowe

Komputery mogą być zastosowane do produkcji filmów animowanych, a także do studiowania efektów plastycznych w innych dziedzinach twórczości — jak architektura czy reżyseria teatralna. Nowe superkomputery wytwarzają obrazy o zadanych właściwościach w czasie akceptowanym przez użytkownika. Reżyser teatralny może dzięki temu sprawdzać wizualnie efekty swoich pomysłów, architekt może przekonać się, jak będzie wyglądał projektowany przezeń obiekt w różnych warunkach. Gdy posługuje się komputerem, może on zmieniać otoczenie budowli, oglądać ją z różnych stron lub przy różnym oświetleniu. Obrazy produkowane przez komputer sprzężony z aparaturą filmową wysokiej jakości odznaczają się doskonałością nieosiągalną w żaden inny sposób.

Konstruowane obrazy mogą przedstawiać sceny realistyczne lub fantastyczne. Można imitować wygląd różnych materiałów, ich fakturę, dobierać odcienie barw, rodzaje oświetleń itd., posługując się metodą prób i błędów. Trudno odpowiedzieć na pytanie, czy komputer stanie się rzeczywiście narzędziem pracy artystów. Na razie wydaje się pewne, że filmy animowane będą produkowane komputerowo. Cena takiego filmu jest porównywalna z ceną filmu realizowanego tradycyjnie, ale komputer jest w stanie wyprodukować dwa pełnometrażowe filmy w ciągu roku, podczas gdy animatorzy wytwórni Walta Disneya, pracując metodami tradycyjnymi, realizują przeciętnie jeden film w ciągu dwóch lat.

Poważnym problemem komputerowej produkcji filmów jest przechowywanie i przesyłanie ogromnej liczby danych. Zakodowanie jednej klatki kolorowego filmu na taśmie 70 mm wymaga przechowania ok. 800 mln bitów informacji. Przesłanie tych danych w celu zarejestrowania pojedynczej klatki na błonie filmowej trwa ok. 10 minut. Dla przyspieszenia przekazywania danych skonstruowano specjalny podsystem wejścia-wyjścia, w którym kanały przekazują dane z prędkością 100 M bajtów/s (dla porównania — znany system DEC UNIBUS przekazuje dane z prędkością 2 M bajty/s). Wspomniany podsystem składa się z pamięci buforowej o pojemności 64 M bajtów oraz trzech procesorów wejścia-wyjścia. Cały podsystem sprzężony jest z aparaturą rejestrującą obrazy na taśmie filmowej.

Jak wygląda produkcja filmu? Najpierw wybiera się obiekty występujące w filmie, a więc figury ludzkie i zwierzęce, przedmioty codziennego użytku itd. Tworzony jest komputerowy obraz każdego przedmiotu — obraz dokładnie taki, jakiego życzy sobie reżyser filmu. Obrazy te przechowywane są w pamięci maszyny, a następnie poddawane są ewolucji wymaganej przez akcję filmu. Na koniec produkowane są kolejne klatki filmu. Reżyser ogląda sceny na monitorze. Jeżeli odpowiadają jego wyobrażeniom, to zapisywane są na taśmie filmowej.

Obecnie maszyna naśladuje metody tradycyjne. Każdy szczegół każdej sceny filmu musi być zaprojektowany przez człowieka. Twórcy filmów komputerowych zapowiadają, że ten stan rzeczy ulegnie zmianie w ciągu paru lat. Reżyser będzie określał obiekty pojawiające się w filmie oraz sposób, w jaki obiekty te mają na siebie oddziaływać. Reszta będzie należała do komputera. Tylko subtelności zachowań ludzkich być może nie poddadzą się tej technice. Na razie istnieją algorytmy do przedstawiania silnych emocji i przesadnych reakcji.

BARIERY ROZWOJU

Mimo swych niewątpliwych zalet, „superkomputeryzacja" napotyka na bariery i rozwija się wolniej niż byłoby to możliwe. Jedną z tych barier, wykazywaną już w podanych przykładach zastosowań, jest problem zaprojektowania algorytmu rozwiązania efektywnie wykorzystującego dany superkomputer.

W oprogramowaniu superkomputerów stosuje się najbardziej rozpowszechniony wśród potencjalnych użytkowników język FORTRAN. Nie jest to jednak klasyczny FORTRAN, ale jego modyfikacja, umożliwiająca proste zapisanie równoległych operacji wektorowych. Również kompilatory tego języka są ukierunkowane na wyszukiwanie równoległości w programie, które skojarzone z właściwościami „superprocesora" przyspieszą uzyskanie wyników. Problem polega jednak na tym, że zapisywane w języku algorytmy muszą być w swej strukturze zupełnie inne od dotychczas stosowanych algorytmów sekwencyjnych. W algorytmie równoległym nie optymalizuje się bowiem liczby operacji jednostkowych, a tylko — jak w przypadku superkomputerów wektorowych — określa formuły operujące całymi wektorami, częściami macierzy itp. Z drugiej strony — pomimo zastosowania języka wysokiego poziomu — algorytm jest uzależniony od rzeczywistej architektury superkomputera. Przykładowo, rozbudowa liczby rejestrów wektorowych z 8 do 16 może zmusić do radykalnych zmian zapisu algorytmu w języku — aby możliwe było efektywne wykorzystanie tego usprawnienia. A już chęć zmiany typu superkomputera pociąga przeważnie za sobą konieczność zaprojektowania nowego algorytmu.

Oczywiście, należy sobie też uświadomić istnienie klasycznego problemu, jakim jest uzyskanie programu poprawnego, pozbawionego błędów. Wiemy ile wymaga to wysiłku i wielokrotnych testowań, nawet dla prostych zagadnień. Tutaj trudności piętrzą się do kwadratu, a nawet sześcianu. Brak wieloletnich doświadczeń programistów oraz nieznajomość nowych architektur superkomputerów przez specjalistów, którzy muszą sformułować podstawowe zależności w rozwiązaniu, prowadzą do tego, że nowe maszyny pracują jeszcze w wielu przypadkach w połowie mocy.

Mimo tych i innych barier, specjaliści sądzą, że rynek superkomputerów będzie się rozrastał, że w latach 1985—1990 będą dostępne na większości uniwersytetów w USA i że ich wykorzystywanie będzie tańsze i efektywniejsze. I oczywiście — zawsze znajdą się problemy, których rozwiązanie będzie wymagało jeszcze szybszych maszyn. A więc CRAY-n ($n \to \infty$) będzie liczył pętlę nieskończoną w czasie — powiedzmy — dwóch minut.

Opracowali

BENIAMIN GOŁDYS
WACŁAW ISZKOWSKI

na podstawie: Proceedings of the IEEE, styczeń 1984; An Interview with E. F. Hoppes, CACM 1984, Vol. 27/4; Computer Humor, CACM 1984, Vol. 27/4

Książki NRD z mikroinformatyki

W sytuacji malejącego dopływu literatury fachowej ze strefy dolarowej, warto zwrócić uwagę na znacznie łatwiej dostępne publikacje NRD. Ze względu na wzrastające zainteresowanie mikroinformatyką, podajemy tytuły wydanych w 1984 r. książek z tej dziedziny przez czołowe wydawnictwa NRD.

Wydawnictwo „Verlag Technik":

- **Bindmann W.: Mikroelektronik. English-Deutsch/Deutsch-English (Technik-Wörterbuch)**
- **Boon K. L.: BASIC für Tischrechner**
- **Franke K.: Einführung in die Mikrorechentechnik**
- **Kieser H., Meder M.: Mikroprozessortechnik. Aufbau und Anwendung des Mikroprozessorsystems U 880 (2. Auflage)**

Wydawnictwo „Fachbuchverlag":

- **Kleinstrechner-TIPS. Heft 1, 2.**

Najlepiej zaopatrzonymi źródłami zakupu książek NRD są: księgarnia Ośrodka Kultury i Informacji NRD, Warszawa, Świętokrzyska 18, oraz księgarnia „UNIVERSUS", Warszawa, Belwederska 20/22, gdzie można je również zamawiać. (WK)

CSK—Komputer Studio Kajkowscy

81-505 GDYNIA ORŁOWO ul. Balladyny 3B, tel. 29-00-18

Komputer osobisty może być przydatny niemal na każdym stanowisku pracy. Wymaga jednak odpowiedniego oprogramowania użytkowego. W ramach tego oprogramowania oferujemy zainteresowanym dostawę uniwersalnych pakietów programowych:

BANK DANYCH CSK, TABPLAN CSK, TEKST CSK, TRANSCOM CSK

To doskonałe narzędzia pracy dla każdego. Aby z nich korzystać, nie trzeba być informatykiem! Zupełnie samodzielnie można tworzyć złożone systemy zarządzania przedsiębiorstwem, każdym przedsiębiorstwem; nawet najbardziej specyficzne uwarunkowania nie są przeszkodą.
To jednak jeszcze nie wszystko... Kiedy dotychczasowe problemy łatwo i szybko zostały rozwiązane — pojawiają się zupełnie nowe. Można wtedy bez kłopotów samemu udoskonalić dotychczasowy system!

BANK DANYCH CSK, TABPLAN CSK, TEKST CSK, TRANSCOM CSK

składają się w zakładowe systemy płacowe, osobowe, finansowo-księgowe lub magazynowe. Korzystając z nich, z łatwością można prowadzić planowanie, kalkulacje i sprawozdawczość. Można też sporządzać kosztorysy i oferty, a nawet prowadzić „automatyczną" korespondencję czy redagować dowolne teksty. Można wreszcie skorzystać z już zgromadzonych zasobów na komputerze ODRA (pod nadzorem systemu GEORGE-3), wykorzystując komputer osobisty jako inteligentny terminal — stację lub emulator TTY.

Nowość:

Oferujemy system operacyjny kompatybilny z CP/M 2.2. dla mikrokomputerów ROBOTRON 5120/5130 oraz systemy finansowo-księgowe FK dla dowolnych mikrokomputerów

---

Szczegółowych informacji udziela:

CSK—Komputer Studio Kajkowscy

81-505 GDYNIA ORŁOWO, ul. Balladyny 3B, tel. 29-00-18

Zastosowanie mikroprocesorów w energoelektronice

Do skromnej kolekcji polskich wydawnictw o mikroprocesorach przybyła kolejna pozycja. Jest nią pierwsza książka poświęcona zastosowaniom i dotyczy wykorzystania mikroprocesorów w energoelektronice [1]. Mimo że jest to książka przeznaczona dla inżynierów elektroników, związanych zawodowo z dziedziną energoelektroniki, nie zaszkodziłoby przypomnieć, co to jest za dziedzina i czym są przekształtniki, będące przedmiotem jej zainteresowania [2].

Dobrym miejscem na to przypomnienie byłby rozdział pierwszy, który stanowi tło dla następnych, bardziej szczegółowych rozdziałów. Zawarto w nim podstawowe wiadomości dotyczące możliwości stosowania mikroprocesorów w urządzeniach przemysłowych w energetyce.

W rozdziale drugim przedstawiono elementarne informacje dotyczące budowy i zasad działania mikroprocesorów, jak przypuszczam — w stopniu wystarczającym do zrozumienia pozostałej części książki. Mam jednak wątpliwości, jak wiele tych informacji należy podawać w książkach poświęconych zastosowaniom — ile razy można czytać to samo o mikroprocesorze INTEL 8080? W każdym razie, autorzy na pewno nie powinni tego tematu traktować jako wypełniacza zawartości.

Rozdział trzeci, zajmujący połowę objętości książki, stanowi jej właściwą treść — dotyczy budowy i programowania podzespołów sprzęgających, najczęściej spotykanych w przemysłowych urządzeniach energoelektronicznych. W rozdziale drugim zawarto omówienie elementów składowych układów mikroprocesorowych, natomiast w trzecim przedstawiono sposoby tworzenia kompletnych układów. Podano w nim również przykłady ich programowania — wszystkie dotyczące mikroprocesora INTEL 8080, choć — jak twierdzi autor (zgodnie z prawdą) — nie zmniejsza to możliwości zrozumienia zasad programowania układów opartych na innych mikroprocesorach. W tym rozdziale autor mógł najpełniej przedstawić swoje doświadczenia zdobyte przy badaniu i realizacji takich układów.

Ostatni rozdział zawiera kilka przykładów dotyczących gotowych urządzeń przekształtnikowych ze sterowaniem mikroprocesorowym, które można budować z układów przedstawionych w poprzednich rozdziałach. Sądzę, że jest to słabsza część książki, która w idealnym przypadku powinna zawierać bardziej zdecydowane przedstawienie metodyki projektowania takich urządzeń opartych na mikroprocesorach (wraz z przykładami).

Opublikowanie książki o zastosowaniach mikroprocesorów w tej właśnie dziedzinie przyjąłem z niejakim zdziwieniem. Obserwując przebieg różnych konferencji dotyczących mikroprocesorów nietrudno stwierdzić, że nader rzadko przedstawiane są tam podobne zagadnienia. Tym większe uznanie należy się autorowi za pracę nad książką, którą wyprzedził kolegów specjalistów z innych dziedzin, jak: telekomunikacja, elektronika medyczna itd.

Uwagi terminologiczne, które mi się nasuwają, nie odbiegają od typowych, przedstawianych wielokrotnie na łamach tego miesięcznika, więc tym razem je pominę. Przed wyrażeniem jednej nie mogę się jednak powstrzymać. Czy zamiast **przełączniki pokrętne**, nie lepiej byłoby mówić — **obrotowe?**

Jest pewne, że takich książek dotyczących zastosowań mikroprocesorów, książek o znaczeniu praktycznym, powinno być znacznie więcej. Oczywiście, jaka jest naprawdę jej wartość pod tym względem, rozstrzygną specjaliści pracujący w tej dziedzinie. Ja mogę stwierdzić, że choć jest to dziedzina niezbyt popularna, nie mogłem kupić książki w księgarni.

JANUSZ ZALEWSKI

[1]) A. Wójciak: Mikroprocesory w energoelektronice. Wydawnictwa Naukowo-Techniczne, Warszawa, 1984. Nakład 4800+200 egz., 164 str., cena 100 zł

[2]) Energoelektronika (elektronika mocy) obejmuje zagadnienia teorii i praktycznego stosowania układów przekształcających energię elektryczną napięcia przemiennego na stałe i odwrotnie oraz — napięcia przemiennego na przemienne o innych parametrach i stałego na stałe o innych parametrach. W konsekwencji, układy przekształcające, czyli przekształtniki, dzielą się na cztery podstawowe grupy: prostowniki, falowniki, przekształtniki prądu przemiennego i stałego. Cechą charakterystyczną urządzeń i układów do przekształcania i sterowania energii elektrycznej w układach energoelektrycznych jest zastosowanie elementów półprzewodnikowych, głównie diod i tranzystorów (na podst. Poradnika Inżyniera Elektryka, WNT, Warszawa, 1974).

Słownik terminów związanych z arytmetyką zmiennoprzecinkową

Użytkownik (ang. user) **systemu arytmetyki zmiennoprzecinkowej** — osoba, sprzęt lub program mający dostęp do operacji ze środowiska programowego określonego normą (p. str. 12).

Dwójkowa liczba zmiennoprzecinkowa (ang. binary floating point number) — ciąg bitów złożony z trzech elementów: znaku, wykładnika ze znakiem oraz mantysy; jej wartością jest iloczyn mantysy przez liczbę 2 podniesioną do potęgi wykładnika z odpowiednim znakiem.

Wykładnik (ang. exponent) — element dwójkowej liczby zmiennoprzecinkowej oznaczający potęgę, do której należy podnieść 2, aby obliczyć wartość tej liczby. Czasami nazywa się go **wykładnikiem nieprzesuniętym** (ang. unbiased exponent, signed exponent).

Wykładnik przesunięty (ang. biased exponent) — suma wykładnika oraz stałej (ang. bias) wybranej w ten sposób, aby wartość wykładnika była dodatnia.

Mantysa (ang. significand) — element liczby zmiennoprzecinkowej zawierający jeden znaczący bit (występujący w sposób jawny lub domyślny, lecz uwzględniany w czasie obliczeń) z lewej strony kropki dwójkowej oraz część ułamkową — z prawej strony.

Część ułamkowa (ang. fraction) — pole mantysy położone po prawej stronie kropki dwójkowej.

Normalne zero (ang. normal zero) — liczba, której wykładnik, przybiera wartość minimalną dla danego formatu, a mantysa jest równa zeru. Normalne zero może mieć znak dodatni lub ujemny. Zera nieunormowane występują tylko w formatach rozszerzonych.

Nieznormalizowana (ang. denormalized) — liczba, której wykładnik przybiera wartość minimalną dla danego formatu, pierwszy bit (jawny lub domyślny) jest równy zeru, a ona sama nie jest normalnym zerem. W celu otrzymania liczby nieznormalizowanej należy przesuwać mantysę w prawo aż do skutku, zwiększając jednocześnie wykładnik.

Nieunormowana (ang. unnormalized) — liczba występująca tylko w formacie rozszerzonym, której wykładnik jest

większy od minimalnego dla danego formatu, a jawny pierwszy bit jest zerem. Jeśli mantysa jest zerem, to liczba jest nieunormowanym zerem. W celu otrzymania liczby nieznormalizowanej należy przesuwać mantysę w prawo aż do skutku, zwiększając jednocześnie wykładnik.

Normalizowanie (ang. normalize) — przesuwanie mantysy liczby niezerowej w lewo, z jednoczesnym zmniejszaniem wykładnika, aż pierwszy znaczący bit po kropce dwójkowej stanie się jednością. Wykładnik traktuje się tak, jakby jego zakres nie był ograniczony. Jeśli mantysa jest zerem, to liczba staje się normalnym zerem. Normalizowanie nie zmienia znaku liczby.

Nie-liczba (ang. not a number) — symboliczny obiekt zakodowany w formacie zmiennoprzecinkowym.

Wskaźnik wyjątku (ang. status flag) — zmienna mająca dwa stany: ustawiony i skasowany, które można kopiować lub kasować programowo. Ustawiony wskaźnik może zawierać dodatkowe informacje zależne od systemu i niekoniecznie dostępne dla programu. Operacje opisane w tej normie mogą jako skutek uboczny ustawiać wskaźniki niedokładnego wyniku, niedomiaru, nadmiaru, dzielenia przez zero i nielegalnej operacji.

Przeznaczenie (ang. destination) — miejsce, do którego każda operacja przekazuje swój wynik, określone w sposób jawny przez użytkownika lub też w sposób niejawny przez system (np. wyniki pośrednie w wyrażeniach arytmetycznych lub argumenty procedur). W niektórych językach wyniki obliczeń pośrednich są przesyłane poza kontrolą programu, jednak norma określa wynik operacji w zależności od formatu przeznaczenia i wartości argumentów.

Tryb (ang. mode) — zmienna, która w celu kontroli przebiegu operacji arytmetycznych może być odczytywana, ustawiana, przechowywana i odtwarzana programowo. Trybem domyślnym jest tryb przyjmowany przez program, chyba że do programu lub jego specyfikacji włączona została instrukcją zmiany trybu. W normie przewidziano następujące tryby:
— **rzutowy** lub **afiniczny**, co dotyczy interpretacji ∞
— **kierunku zaokrąglania**, co dotyczy kierunku błędów zaokrągleń
— **zaokrąglania** (w niektórych implementacjach) — w celu zmniejszenia precyzji wyników pośrednich
— **normalizacji** lub **ostrzegania**, co dotyczy uwzględniania wyników z niedomiarem
— **dozwolonych** lub **zabronionych przerwań programowych** — w celu obsługi wyjątków.

ZBIGNIEW BANASIK

Ceny ogłoszeń

Od 1 stycznia br. obowiązują następujące ceny ogłoszeń publikowanych na naszych łamach:
ogłoszenia duże (zależnie od objętości):
cała strona — 35 tys. zł, 3/4 — 30 tys., 1/2 — 25 tys., 1/4 — 20 tys., 1/8 — 15 tys.
ogłoszenia drobne (zależnie od liczby słów):
jedno słowo — 30 zł
Dodatki do ceny podstawowej:
— za dodatkowy kolor (na okładce) +30%
— za zamieszczenie ogłoszenia na czwartej stronie okładki +100%
— za zamieszczenie ogłoszenia na trzeciej stronie okładki +50%.
Zniżki:
— za ogłoszenie 3—5-krotne —5%
— za ogłoszenia 6—10-krotne —10%
— za ogłoszenia 11-krotne i powyżej —20%
— za artykuły reklamowe i wkładki wykonane przez zleceniodawcę —40%
— za bloki i biuletyny wykonane przez zleceniodawcę — maks. —60%.

W przypadku dostarczenia przez zleceniodawcę materiału ilustracyjnego nie odpowiadającego warunkom technicznym druku lub tekstu wymagającego redakcyjnego opracowania, do powyższych cen doliczane będą koszty odpowiednich usług fotograficznych, graficznych lub przygotowania tekstów.

Kott R. K.: BASIC dla mikrokomputerów — ewolucja języka

INFORMATYKA 1985, nr 1, s. 2

Charakterystyka aktualnego stanu rozwoju języka BASIC, modyfikowanego pod kątem specyficznych potrzeb użytkowników mikrokomputerów. Omówiono nowe instrukcje i funkcje języka.

Krępski A.: PASCAL MT+ — dialekt języka PASCAL dla mikrokomputerów

INFORMATYKA 1985, nr 1, s. 6

Charakterystyka języka programowania PASCAL MT+ przeznaczonego dla użytkowników mikrokomputerów. Omówiono pakiet programów narzędziowych SPP oraz kompilator tego języka.

Wilczyński W.: PLZ — język programowania mikroprocesorów (2)

INFORMATYKA 1985, nr 1, s. 9

Druga część charakterystyki języka programowania mikroprocesorów PLZ, zawierająca omówienie ważniejszych instrukcji, struktury programu oraz zasad łączenia modułów.

Banasik Z.: Projekt normy IEEE — arytmetyka zmiennoprzecinkowa

INFORMATYKA 1985, nr 1, s. 12

Omówienie projektu normy dotyczącej ujednolicenia formatu liczb zmiennoprzecinkowych. Projekt opracował Komitet ds. Normalizacji Arytmetyki Zmiennoprzecinkowej powołany przez amerykańską organizację IEEE.

Котт Р. К.: BASIC для микро-ЭВМ — эволюция языка

INFORMATYKA 1985, № 1, стр. 2

Характеристика актуального состояния развития языка BASIC, модифицированного с точки зрения специфических потребностей пользователей микро-ЭВМ. Обсуждены новые операторы и функции языка.

Кремпски А.: PASCAL MT+ — диалект языка PASCAL для микро-ЭВМ

NFORMATYKA 1985, № 1, стр. 6

Характеристика языка программирования PASCAL MT+ предназначенного для пользователей микро-ЭВМ. Обсуждены пакет вспомогательных программ SPP и компилятор этого языка.

Вильчински В.: PLZ — язык программирования микропроцессоров (2)

INFORMATYKA 1985, № 1, стр. 9

Вторая часть характеристики языка программирования микропроцессоров PLZ, содержащая обсуждение важнейших операторов, структуры программы и прамил соединения модулей.

Банасик З.: Проект станрарта IEEE — арифметика с плавающей запятой

INFORMATYKA 1985, № 1, стр. 12

Обсуждение проекта стандарта касающегося унификации формата чисел с плавающей запятой. Проект разработан Комитетом стандартизации арифметики с плавающей запятой, созданным американской организацией IEEE.

Kott R. K.: BASIC for microcomputers — evolution of the language

INFORMATYKA 1985, No. 1, p. 2

Characteristics of actual development of BASIC, which is modified for specific needs of microcomputer users. New statements and functions of the language are discussed.

Krępski A.: PASCAL MT+ — a PASCAL dialect for microcomputers

INFORMATYKA 1985, No. 1, p. 6

Characteristics of PASCAL MT+ language, which is appropriated for microcomputer users. SPP utility program package and the language compiler are discussed.

Wilczyński W.: PLZ — a programming language for microprocessors (2)

INFORMATYKA 1985, No. 1, p. 9

Second part of the PLZ microprocessor programming language which includes presentation of some more important statements, program's structure and rules for module connection.

Banasik Z.: IEEE standard proposal for floating point arithmetics

INFORMATYKA 1985, No. 1, p. 12

Presentation of standard proposal for unified format of floating point numbers. The proposal is elaborated by the Committee for Floating Point Arithmetics Standarization, which is created by IEEE.

Kott R. K.: BASIC für Mikrorechner — eine Evolution der Sprache

INFORMATYKA 1985, Nr. 1, S. 2

Eine Charakteristik der aktuellen Entwicklung von BASIC-Sprache, die für Mikrorechnerbenutzer spezifische Bedürfnisse modifiziert wurde. Es wurden neue Anweisungen und Funktionen der Sprache vorgestellt.

Krępski A.: PASCAL MT+ — ein PASCAL-Dialekt für Mikrorechner

INFORMATYKA 1985, Nr. 1, S. 6

Eine Charakteristik der PASCAL MT+ -Sprache, die für Mikrorechnerbenutzer bestimmt ist. Es wurden SPP-Dienstprogrammpaket und Compiler der Sprache vorgestellt.

Wilczyński W.: PLZ — eine Programmiersprache für Mikroprozessoren (2)

INFORMATYKA 1985, Nr. 1, S. 9

Zweiter Teil einer Charakteristik von der PLZ-Sprache für Mikroprozessorenprogrammierung, die Besprechung der wichtigeren Anweisungen, Struktur des Programmes und Grundsätze für Modulverbindung umfasst.

Banasik Z.: IEEE-Norm-Entwurf für Gleitkommaarithmetik

INFORMATYKA 1985, Nr. 1, S. 12

Eine Besprechung des Normentwurfes für Formatvereinheitlichung der Gleitkommazahlen. Der Entwurf wurde von IEEE-Komitee für Gleitkommarithmetiknormung bearbeitet.

Zasady prenumeraty

Zamówienia i przedpłaty na prenumeratę INFORMATYKI przyjmuje Zakład Kolportażu Wydawnictwa NOT SIGMA. Adres pocztowy: Wydawnictwo NOT SIGMA — Zakład Kolportażu, 00-950 Warszawa, skr. poczt. 1004. Konto bankowe: III O/M NBP Warszawa nr 1036-7490-139-11.

PRENUMERATORZY ZBIOROWI — jednostki gospodarki uspołecznionej, instytucje i organizacje — składają zamówienia w formie wpłaty-przelewu na specjalnym blankiecie opracowanym przez Wydawnictwo. Część tego blankietu zawiera listę tytułów czasopism kolportowanych przez Wydawnictwo.

W przypadku większej liczby odbiorców:
— od 2 do 5 — należy dokonać wpłaty oddzielnie dla każdego odbiorcy
— od 6 odbiorców — dokonać jednej wpłaty, z powołaniem się na znak kancelaryjny pisma przesłanego do Zakładu Kolportażu, a zawierającego wykaz adresów poszczególnych odbiorców i zamawianych dla nich tytułów czasopism oraz numer konta bankowego, z którego dokonany został przelew. Na wpłacie-zamówieniu należy podać łączne liczby egzemplarzy poszczególnych tytułów czasopism.

WPŁATY-ZAMÓWIENIA przyjmowane są w terminach:
- **do 28 lutego — na II, III i IV kwartał**
- **do 31 maja — na II półrocze i IV kwartał**
- **do 31 sierpnia — na IV kwartał**

PRENUMERATA STAŁA-WIELOLETNIA dotyczy tylko prenumeratorów zbiorowych. Zamawiający będzie otrzymywał z Wydawnictwa potwierdzenie kontynuacji prenumeraty wraz z wezwaniem do zapłaty. Zmiany w prenumeracie należy zgłaszać pisemnie w terminach obowiązujących dla składania zamówień-wpłat.

PRENUMERATORZY INDYWIDUALNI — osoby fizyczne — zamawiają prenumeratę dokonując wpłaty na blankiecie opracowanym przez Wydawnictwo NOT SIGMA (dostępnym w urzędach pocztowych) lub na blankiecie przekazu NBP — w terminach j.w. Na odwrocie środkowego odcinka przekazu, przeznaczonego dla adresata-posiadacza rachunku (Wydawnictwa), należy podać tytuł zamawianego czasopisma, okres prenumeraty i liczbę egzemplarzy.

Ceny INFORMATYKI

Cena egz.	Prenumerata w złotych					
	normalna			ulgowa		
	kwart.	półrocz.	roczna	kwart.	półrocz.	roczna
100,—	300,—	600,—	1200,—	105,—	210,—	420,—

PRENUMERATA ze zleceniem wysyłki za granicę jest dwukrotnie droższa.

PRENUMERATA ULGOWA. Uprawnieni są do niej: indywidualni członkowie Stowarzyszeń Naukowo-Technicznych, studenci oraz uczniowie szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu przekazu (na odwrocie środkowego odcinka) pieczęcią Koła SNT NOT, wyższej uczelni lub szkoły.

UWAGA!

1. **Instytucję zainteresowane prenumeratą naszego czasopisma mogą otrzymać cennik i blankiet-zamówienie w Zakładzie Kolportażu oraz w:**
   - **— Biurach Wydawniczych:**
     90-020 Łódź, Pl. Komuny Paryskiej 5a
     50-019 Wrocław, ul. Świerczewskiego 74
   - **— Oddziale Wydawnictwa:**
     40-014 Katowice, ul. Dąbrowskiego 23
   - **— Oddziałach Wojewódzkich NOT**
   - **— Zarządach Głównych Stowarzyszeń Naukowo-Technicznych**
   - **— Redakcjach czasopism branżowych.**
2. **Powyższe warunki prenumeraty dotyczą czasopism kolportowanych przez Wydawnictwo NOT SIGMA.**

Dodatkowych informacji o prenumeracie udziela Zakład Kolportażu, tel. 40-00-21 wew. 293, 299 lub 40-35-89.

EGZEMPLARZE ARCHIWALNE czasopism wydawanych przez Wydawnictwo NOT SIGMA (od 1977 roku) można zamawiać w Zakładzie Kolportażu, 00-950 Warszawa ul. Mazowiecka 12 (tel. 26-80-16) lub nabywać w Klubie Prasy Technicznej, 00-950 Warszawa, ul. Mazowiecka 12 (tel. 27-43-65).

Grafika w INFORMATYCE

Niedawno (INFORMATYKA 8/84) ogłosiliśmy konkurs na najlepszą grafikę komputerową. Ponieważ byłem jednym z pomysłodawców tego konkursu, postaram się sprecyzować na jakiej grafice nam zależy. Dostaliśmy bowiem m.in. rysunek Miki Mouse wykonany na drukarce wierszowej. Rzecz nie w tym, iż jest to naiwne naśladownictwo rysunków Walta Disneya, ani też w niewątpliwej brzydocie tego wydruku. Po prostu — od komputerów (czyli od programistów) oczekujemy czegoś więcej.

Na następnej stronie publikujemy kolejną grafikę, która — jak się nam wydaje — dobrze spełnia warunki stawiane grafice komputerowej. Na marginesie dodam, iż pomysł zamieszczonego rysunku pochodzi od R. P. Saundersa (Computer Journal, 1972, nr 15, str. 160—169).

Oto wydruk naszego programu. Może ktoś narysuje podobną grafikę, ale w większym zakresie zmian x i y?

```
      PROGRAM KWIAT
      III=120
      XL=0.
      YL=0.
      YH=3.*FLOAT(III)
      IIJ=1.5*III
      XH=3.*FLOAT(IIJ)
      CALL PLOTER(XL,XH,YL,YH)
      DO 40 I=1,III
      DO 30 J=1,IIJ
      Y=I
      X=J
      Z=X/Y
      Z=Z-INT(Z)
      IZ=Z*1000.
      IQ=IZ-10*(IZ/10)
      IF (IQ) 101,102,103
  101 WRITE(5,1000)
      STOP
  103 L=0
      DO 10 KI=1,3
      YY=FLOAT(3*I+KI)
      XX=3*J
      CALL MOVE(XX,YY)
      DO 20 KJ=1,3
      L=L+1
      IF (L.GT.IQ) GOTO 20
      XX=FLOAT(3*J+KJ)
      CALL DRAW(XX,YY)
   20 CONTINUE
   10 CONTINUE
  102 CONTINUE
   30 CONTINUE
   40 CONTINUE
 1000 FORMAT(6H ERROR)
      STOP
      END
```

Grafika komputerowa winna przede wszystkim wykorzystywać sposoby obrazowania właściwe tylko komputerom. Skoro komputery są rodzajem skomplikowanych liczydeł, to możemy oczekiwać, iż przedstawią nam pewne zależności między liczbami. Nasza grafika pokazuje wartość trzeciej cyfry po przecinku wyniku dzielenia współrzędnej X rysunku przez współrzędną Y — praca programisty ograniczyła się tu do zauważenia, że ta trzecia cyfra może przyjmować wartości między 0 i 9, a zatem kwadrat o wymiarach 3×3 częściowo zaczerniany łatwo je zobrazuje (pamiętajmy, że grafika nie musi być czarno-biała, choć na razie INFORMATYKA może reprodukować tylko takie rysunki...).

Wyjaśniwszy metodę konstrukcji naszej grafiki, pozwólmy sobie zauważyć, że nikt prawdopodobnie nie odgadłby reguły, jaka wykreowała tak niesamowity obrazek! To właśnie druga cecha dobrej grafiki komputerowej: możliwość przedstawienia zależności, z których bez pomocy komputera nigdy byśmy sobie nie zdawali sprawy. Niektórzy ten dział informatyki nazywają matematyką eksperymentalną. Co więcej — wydaje się, że niektóre działy analizy matematycznej (np. analiza funkcji zespolonych) będą łatwiejsze do nauczenia, jeżeli studenci będą mogli z o b a c z y ć funkcje, które dotąd przedstawiano tylko jako szeregi (proste pytanie: ilu Czytelników INFORMATYKI potrafi wyobrazić sobie funkcję: |sin(z)|*)?!.

Dochodzimy do sedna sprawy: grafika komputerowa tak naprawdę jest wykonywana przez komputer, ale przez to nie zmniejszają się przecież wymagania co do inwencji człowieka, który go programuje. Jak w każdej dziedzinie sztuki, rola człowieka jest kreująca — choć mechaniczne wykonanie pomysłu jest tu powierzone komputerowi. Na marginesie dodam, że sam proces rysowania może trwać o wiele dłużej niż wymyślenie lub wyszukanie dobrego pomysłu. Nasz pomysł zasugerowała praca „Pattern Recognition" (ed. B. G. Batchelor, Plenum Press, 1976), a więc książka zajmująca się rozpoznawaniem obrazów, a nie ich konstrukcją! Ploter (pisak X-Y) dołączony poprzez konwerter cyfrowo-analogowy, którym dysponujemy, rysował obrazek ponad jedenaście (!) godzin i to wcale nie dlatego, że maszyna MERA 400 jest wolna.

Chcielibyśmy, by Czytelnicy INFORMATYKI przesyłali nam swoje rysunki, które będą spełniały warunek kreacyjności. Chcielibyśmy też, by rysunki te były intrygujące, ciekawe, po prostu — inne niż grafika rysowana przez plastyków. Mamy nadzieję, że wśród polskich informatyków wrażliwość na piękno i dobre pomysły pójdą w parze.

Ogłaszamy nieustający konkurs na najlepsze prace graficzne. Nagrody skromne: publikacja na czwartej stronie okładki i ...honorarium.

Dopuszczamy wszelkie metody kreowania rysunków: wydruki komputerowe, zdjęcia ekranów monitorów, rysunki z ploterów. Jeżeli pomysł grafiki został zaczerpnięty z jakiegoś źródła, prosimy o podanie odnośników i ewentualnie własnych komentarzy (jak wykonano rysunek, czas wykonania, sposób kreślenia itp.). Jeżeli ktoś nie ma możliwości wykonania samodzielnie grafiki, prosimy o przysyłanie pomysłów — postaramy się w miarę naszych możliwości wykreślić „zadany" wzór. Wydaje się jednak, że najlepszym sposobem sprawdzenia atrakcyjności pomysłu jest zobaczenie uzyskanych kształtów. Gdy w teczce redakcyjnej uzbieramy więcej ciekawych grafik, być może uda się nam wydać specjalny numer pisma poświęcony grafice.

Czekamy na przyszłych Piccassów komputerowych!

JAKUB TATARKIEWICZ

*) Rzeźba funkcji $f = \sqrt{\sin^2 x + \sin h^2 y}$

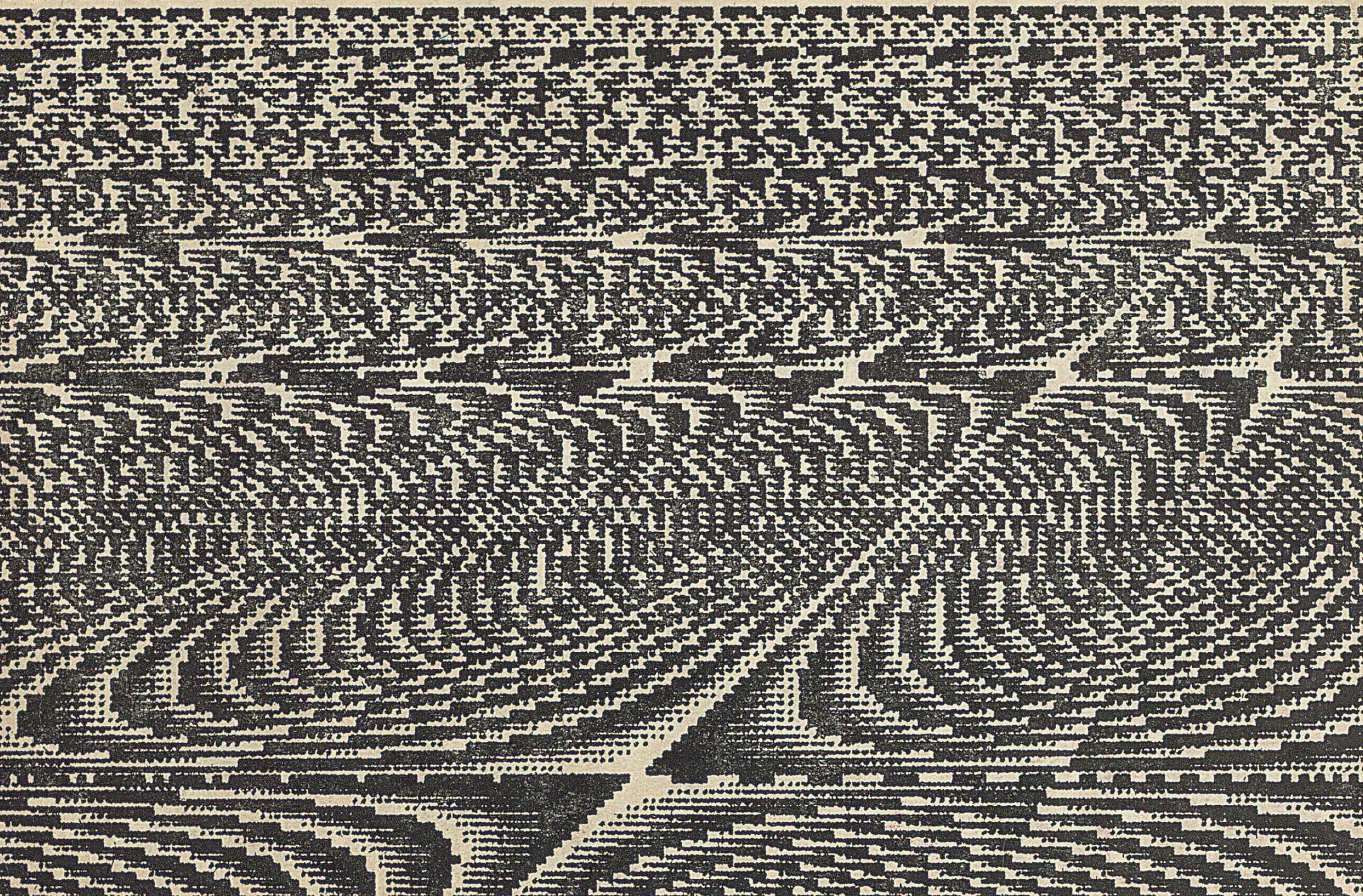
Kalendarz 1985

W kalendarzu zapowiadamy tylko te imprezy, o których zostaliśmy poinformowani bezpośrednio przez organizatorów. Podajemy tylko główne informacje — szczegółowe otrzymać można w redakcji.

WIOSNA

- HARDWARE AND SOFTWARE COMPONENTS AND ARCHITECTURES FOR THE 5th GENERATION — Paryż, 5—7 marca, organizator: AFCET INFORMATIQUE
- TAPSOFT, International Joint Conference on Theory and Practice of Software Development — Berlin Zachodni, 25—29 marca, organizatorzy: TU Berlin, GI, EATCS
- MIKRONIKA-85, Postęp w budowie precyzyjnego sprzętu elektroniczno-mechanicznego — Warszawa, 15—16 maja, organizatorzy: SIMP, Politechnika Warszawska
- SM 85 — Warszawa, 21—24 maja, organizator: Klub Użytkowników Minikomputerów SM, Polskie Towarzystwo Informatyczne
- PROLAMAT 1985, Software for Discrete Manufacturing — Paryż, 11—13 czerwca, organizator: AFCET

LATO

- IX KRAJOWA KONFERENCJA AUTOMATYKI, Stan i perspektywy automatyki w Polsce — Łódź, koniec czerwca, organizatorzy: Polski Komitet Pomiarów i Automatyki NOT, Politechnika Łódzka

RAFAŁ PIETRAK JAKUB TATARKIEWICZ